Project Number: CS-MXC-1041

Crypto-SmartLock: Applying Cryptography to Physical Security

A Major Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Kevin J. D'Aquila | **Frank L. Gerratana**

Anthony P. Oteri | **Jeffrey A. Rosenberger**

Date: March 10, 2004

Approved:

Professor Michael Ciaraldi, Major Advisor

Professor R. James Duckworth, Co-Advisor

1. Cryptography
2. Security
3. Smart Card

Abstract

Electronic locks are rapidly replacing traditional mechanical locks in today's organizations, as they are easier to manage and are perceived as more secure. However, most electronic systems do not provide adequate security and are often even less secure than their mechanical counterparts. This project combines the security of modern cryptography and flexibility of Smart Cards to create an electronic door lock system that provides greater security than a mechanical lock but still allows the conveniences granted by an electronic system.

We would like to thank our advisors, Professor Ciaraldi and Professor Duckworth, Scott Longley for interface feedback and for putting up with living with us through this project, Valerie Galluzzo for her support, Dave Ludwig for his help debugging, John Waymouth for his help with LaTeX and the ECE shop for supplying components and the custom-made enclosure.

Contents

1 Introduction **8**
- 1.1 Concepts . . . 8
- 1.2 Project Goals . . . 9
- 1.3 Components of a Typical Electronic Door Lock . . . 9
- 1.4 Layout of the Report . . . 10

2 Background **11**
- 2.1 Existing Systems . . . 11
  - 2.1.1 Messerschmitt Classic/Executive . . . 11
  - 2.1.2 Bosch Easikey/ReadyKey . . . 12
  - 2.1.3 Id iClass . . . 12
  - 2.1.4 AdelLock . . . 13
  - 2.1.5 MaxKing MaxLock . . . 13
  - 2.1.6 BEST Basis V . . . 14
  - 2.1.7 VingCard . . . 14
  - 2.1.8 TimeLox . . . 14
- 2.2 Target Market . . . 14
  - 2.2.1 Possible Markets . . . 17
- 2.3 Chosen Target Market . . . 18
- 2.4 Cryptography . . . 19

3 System Requirements **21**
- 3.1 End User Requirements . . . 21
- 3.2 Overall System Requirements . . . 22
- 3.3 Cryptography and Security Requirements . . . 22

4 Possible Methods of System Operation **24**
- 4.1 Access Methods . . . 24
- 4.2 Key Management Interface . . . 25
  - 4.2.1 Key Addition Methods . . . 25
  - 4.2.2 Key Removal Methods . . . 26
- 4.3 Overall System Paradigm . . . 26
  - 4.3.1 Connected vs Disconnected . . . 26
  - 4.3.2 Centralized vs Decentralized . . . 27

4.3.3 A Disconnected, Centralized System 27
4.4 Chosen Method of Operation 27

5 Design Alternatives 29
5.1 Physical Interfaces 29
5.1.1 Criteria 29
5.1.2 Possibilities 29
5.2 Smart Cards 31
5.2.1 Proprietary Smart Cards 32
5.2.2 "Blank" Smart Cards and Designing an OS From Scratch 32
5.2.3 BasicCards 33
5.3 Microcontrollers 34
5.3.1 Criteria 34
5.3.2 Possibilities 35
5.3.3 MSP430 36
5.3.4 MSP430 Development Systems 36

6 System Design and Specifications 37
6.1 Communications Protocol 37
6.1.1 Communication Elements 37
6.1.2 Cryptographic Keys 38
6.1.3 Commands 38
6.1.4 System Component Interaction 40
6.1.5 Communication Internals 40
6.1.6 Protocol Rationale 44
6.2 Key Management System Hardware 46
6.3 Key Management System Software 46
6.3.1 Human Interface Design 47
6.3.2 Class Hierarchy 53
6.3.3 Data Model 54
6.3.4 Command Processing 55
6.4 BasicCard Hardware 57
6.5 BasicCard Software 58
6.5.1 BasicCard Commands used by both Door and Key Management System 59
6.5.2 BasicCard Commands used by the Door 59
6.5.3 BasicCard Commands used by the Key Management System 60
6.6 Door Hardware 65
6.6.1 Human Interface Design 65
6.7 Door Software 67
6.7.1 Cryptography on the Door 67
6.7.2 Serial Communication 68
6.7.3 Flash Memory 68
6.8 System Flow 69
6.8.1 Adding a user or door to the system 69
6.8.2 Command Passing 69

6.9 Attack Scenarios and Prevention . . . 75
6.9.1 Attacks Prevented . . . 75
6.9.2 Attacks Not Prevented . . . 79

7 Production Design **80**
7.1 Door . . . 80
7.1.1 Components . . . 80
7.1.2 Power . . . 81
7.1.3 Cost . . . 83
7.1.4 Block Diagram . . . 84
7.2 BasicCard . . . 85
7.3 Key Management System . . . 85

8 Conclusion **87**
8.1 Future Work . . . 87
8.1.1 Random Numbers . . . 87
8.1.2 Smart Card Evolution . . . 88

A User Manual **89**
A.1 Crypto-SmartLock System Requirements . . . 89
A.2 Basic Key Management System Operation . . . 89
A.2.1 Adding Records of Users . . . 90
A.2.2 Adding Records of Doors . . . 90
A.2.3 Granting and Denying Users Access to Doors . . . 91
A.2.4 Updating Door Lock Modules . . . 91
A.2.5 Putting Commands on Key Cards . . . 92
A.3 Door Lock Modules . . . 93
A.3.1 Setting up a Door Lock Module for the First Time . . . 94
A.4 Removing Users and Doors from the System . . . 95
A.5 Reissuing Lost Commands . . . 95

B Table of Contents for CD **97**

C Bill of Materials **98**

D Development Tools **99**
D.1 MSP430 Compilers . . . 99
D.2 BasicCard . . . 99
D.3 Visual Studio®.NET 2003 . . . 100

E Internal Administrative Tasks **101**
E.1 Web Site . . . 101
E.2 CVS . . . 101

Bibliography **102**

Index **104**

List of Figures

2.1 Messerschmitt Door Mounted Locks 12
2.2 Messerschmidt Wall Mounted Locks 12
2.3 ID iClass Retrofitted Chip 13
2.4 AdelLock System 13
2.5 MaxLock Mounted in Wood 14
2.6 Best Basis V Lock 15
2.7 Ving Card System 16
2.8 TimeLox System installed in door 17

4.1 Overall System Block Diagram 28

6.1 Door-Card Interaction Diagram 41
6.2 Card-System Interaction Diagram 42
6.3 Manage Users Tab Screenshot of the GUI 47
6.4 Manage Doors Tab Screenshot of the GUI 48
6.5 Manage Access Tab Screenshot of the GUI 49
6.6 Reissue Commands Tab Screenshot of the GUI 50
6.7 Assign Commands Tab Screenshot of the GUI 52
6.8 Card Reader Tab Screenshot of the GUI 52
6.9 Key Management System Class Hierarchy 53
6.10 Prototype Block Diagram 62
6.11 Crypto-SmartLock Prototype - Outside View with JTAG interface 63
6.12 Crypto-SmartLock Prototype - Outside View with Power Connector and Switch 63
6.13 Crypto-SmartLock Prototype - Inside 64
6.14 Crypto-SmartLock Prototype - Bottom with Rubber Feet 64
6.15 Prototype Schematic 66
6.16 Flowchart for adding a user to the system 70
6.17 Flowchart for adding a door to the system 71
6.18 Flowchart for command passing 72
6.19 Flowchart for commands on BasicCard 73
6.20 Flowchart for commands on door 74

7.1 Current draw during unlock process 82

7.2 Power Requirements in mAh for a one year replacement cycle . . 83
7.3 Production Design Block Diagram 84

A.1 The Crypto-SmartLock Key Management System 90
A.2 The Crypto-SmartLock Prototype Door Module 93

List of Tables

2.1 Existing System Features 11

5.1 ZeitControl BasicCard Enhanced - Currently Available Cards . . 33
5.2 ZeitControl BasicCard Professional - Currently Available Cards . 34
5.3 Development Board Comparison 36

6.1 Byte Structure of Door Commands 40
6.2 Byte Structure of Command Headers 43
6.3 Byte Structure of Confirmations 43
6.4 Byte Structure of Confirmation Headers 43
6.5 Byte Structure of Authorization Key Information 44
6.6 System Requirements for Key Management System 46
6.7 Contents of Confirmation 59
6.8 Put Command Contents 60
6.9 Set Key Contents 61
6.10 Setup Card Contents 61
6.11 Door Lock Human Interface Design 65

7.1 Current Draw of Components for Production System 81
7.2 Current draw during unlock process 81
7.3 Power Usage 82
7.4 Costs of Production Design 83

A.1 Door Lock Human Interface Design 94

D.1 MSP430 Compilers 99

Chapter 1

Introduction

1.1 Concepts

In the United States there are many companies, hotels, and educational institutions which have recently invested a large amount of time and money into replacing mechanical locks with magnetic stripe card-reader locks. Typically these cards contain a simple user ID number which is plainly transmitted to the lock as a means of authentication. For a small amount of money a malicious user could purchase a programmer and both read and re-program cards to trick the door into thinking they are a trusted user. It is also common practice to assign these numbers using a social security number, employee ID number, or a card ID number. The problem with this approach is that these numbers are easily obtainable, and are usually printed on the physical card.

Cryptography or "secret writing" has been around since before Caesar had his empire, and is a method of scrambling the parts of a message in such a way that they no longer will make sense to a casual observer. One of the simplest cryptographic ciphers is known as the "Shift Cipher" in which all of the letters in the alphabet are rotated by a fixed number of positions. Major advances in computing power and mathematics have long since deprecated the shift cipher, and a new standard has been introduced known as the Advanced Encryption Standard (AES).

Smart Cards were designed as a replacement to magnetic-stripe cards. They are similar in size to a credit card, and contain a micro-processor and a small amount of memory. Unauthorized extraction of data from a Smart Card is much more difficult than from a magnetic-stripe card because Smart Cards were designed with data-security in mind. Currently, Smart Card technology is more widespread in Europe than it is in the United States. This situation exists because Europe's telecommunication infrastructure was not set up to provide the networking services needed for a magnetic strip card based credit processing system. Because this system has been used successfully in the US for many years, there is reluctance to move to Smart Cards, while Europe, lacking this

existing system, has been much quicker to embrace Smart Card technology.[20]

Modern card-key systems provide more flexible security opportunities than do physical brass-key locks. By centralizing electronic card-key system, it is possible for security personnel to monitor accesses to a door to prevent unauthorized accesses from a central location. Another factor in consideration for electronic card-key systems is long-term costs. Physical locks require a minimum initial investment to use, but replacing keys is expensive, and when a key is lost, the locks and all keys must be replaced. Electronic locks require a slightly larger initial investment, but lost cards only need be deactivated and replaced. This saves the costly, time consuming, task of replacing all locks and keys imposed by mechanical locking systems.

1.2 Project Goals

As a team, we plan to bring concepts from cryptography into use in physical security by implementing a door lock that uses cryptography to authenticate users to the door. Because we decided that our system was to be composed of unconnected, stand-alone door locks, we needed to determine exactly how messages were to be transmitted between doors and a centrally-located key management database system. We attacked this communication problem by designing a cryptographic network protocol using Smart Cards to transmit messages. Finally we needed to design the centrally-located key management database system to be able to generate commands, issue cards, keep track of all doors and users, and receive acknowledgments from the doors.

The goal of this project is to integrate cryptography into physical security by developing a Smart Card based door lock and software based key management system based on a cryptographic protocol. The overall system consists of a centralized server for key management and a series of unconnected "stand-alone" door locks. The system will attempt to bridge the gap between connected, centralized systems, and unconnected, stand-alone systems, and to improved the current state of the market by using encryption to authenticate users, and improve the security and the integrity of the data on both the card, and the door.

1.3 Components of a Typical Electronic Door Lock

The typical electronic door lock consists of two to three main components, depending on the design. The first component is the door lock assembly. This is typically either attached to the door or the wall next to the door and will contain a microprocessor and memory to store user access lists as well as some interface to the second main component, the key. The key is usually a small portable card or similar device which every user of the system possesses. For a typical system, the user will generally present, insert, or swipe his or her key

to some form of reader attached to the door lock assembly. The final part of the system is only applicable in a "Centralized" system, and that is the central key management system. Typically this is a requirement of large systems and is a software-based component of the system responsible for managing users and monitoring accesses to the system.

1.4 Layout of the Report

The report is separated into several sections. The first is the background and requirements section. This describes the important background ideas behind an electronic door-lock system and lays the groundwork for the following section dealing with the design and implementation of our prototype. The second section describes in detail the prototype system including the design of the protocol, the complete implementation of our prototype system, and the rationale for the security behind our system. The final section outlines the description of a production system and highlights different ideas for future work.

In the following chapters, the groundwork for the development of our project is described. These chapters give an overview of the concepts, and outline the current state of the market by introducing a variety of existing systems and possible target markets. They also determine the necessary requirements of our prototype system.

Chapter 2

Background

2.1 Existing Systems

We examined several electronic door lock systems on the market today to learn how the market is currently served, and find ways to serve it better. This provided information on what capabilities a new system would have to have, and what features customers might expect. It also allowed us to determine the level of security such systems usually implement.

The existing systems we surveyed are summarized in Table 2.1 and are briefly described in the following sections.

2.1.1 Messerschmitt Classic/Executive

The Messerschmitt systems use magnetic cards, or, as the company heavily promotes, RF-based cards running in HF frequencies.[2] These systems are very centralized, with all key management done at a central server connected to the locks. In the Executive version of the system, authentication is also done

Table 2.1: Existing System Features

System	Smart	Magnetic	RF	Custom	Keypad	Key Management	Lock Power
Messerschmitt	No	Yes	Yes	No	No	Central	Battery/Wired
Bosch Easikey	No	No	No	Yes	No	Central	Wired
Bosch Readykey	No	No	No	No	Yes	Unclear	Wired
ID iClass	Yes(RF)	No	No	No	No	Unclear	Unclear
AdelLock	Yes	Yes	Yes	Yes	Unclear	Unclear	Unclear
MaxKing	Yes	No	No	No	No	Unclear	Battery
BEST Basis V	Yes	Yes	Yes	Yes	Unclear	Portable/PDA	Battery
VingCard	Yes	Yes	No	No	No	Central Software	Battery
Timelox	No	Yes	No	No	Yes	Central Software	Unclear

centrally; the locks are not usable without the central server. The Messerschmitt system is best suited to very large office buildings or hotels, where there is a need for centralized operation of the system, and also a staff available around the clock to ensure the central system is running at all times. Messerschmitt's door mounted lock systems are shown in Figure 2.1. The wall mounted locks are shown in Figure 2.2.

Figure 2.1: Messerschmitt Door Mounted Locks

Figure 2.2: Messerschmidt Wall Mounted Locks

2.1.2 Bosch Easikey/ReadyKey

The Bosch Easikey system uses a custom keychain tag-type device instead of a traditional card or key.[18] These keys are encoded from the factory with a unique serial number. In the production system all data should be stored with a unique code. Key management is done from a central server, which is permanently connected to the locks. The ReadyKey system, on the other hand, is based on a keypad, and the locks do not have to be connected to a central server. In both systems, the locks must be wired to a power source. The Easikey system is suited to a midrange office or campus, whereas ReadyKey is best used for very specific applications, such as closets or safes.

2.1.3 Id iClass

Id Enhancements offers a product based on proximity smart cards.[7] These cards do not require a physical contact, but instead communicate with a reader

with, and are powered by, RF signals. In addition, since there are no dimensional requirements imposed by the reader, they provide the option of attaching the smart card electronic component to an existing identification card as shown in Figure 2.3.

Figure 2.3: ID iClass Retrofitted Chip

2.1.4 AdelLock

The AdelLock system is notable for its flexibility; locks exist that can accept smart cards, magnetic cards, RF devices, and can include a keypad.[9] These locks are suited to an environment where users have a wide range of requirements for their keys. One of the AdelLock systems is shown in Figure 2.4.

2.1.5 MaxKing MaxLock

MaxKing's system is based on Smart Cards, and has limited centralization.[10] If a central server exists, it is used only for monitoring. The locks are stand-alone and battery-powered. The MaxLock is ideal for smaller environments that do not need centralization but could benefit from the security of a Smart Card. The complete lock can be installed in wooden doors, as shown in Figure 2.5.

Figure 2.4: AdelLock System

2.1.6 BEST Basis V

The Basis V is a disconnected centralized system.[17] Locks are battery powered, and can be modified via a PDA or laptop connected directly during regular maintenance/updates. The locks can accept, smart cards, magnetic cards, or RF devices, and can include a keypad for additional security. This variety of options is detailed in Figure 2.6. The Basis V is good for smaller buildings or specialized applications with a limited amount of doors or access points.

2.1.7 VingCard

VingCard's product lines can use either magnetic cards or Smart Cards.[21] One of VingCard's door interfaces is shown in Figure 2.7. The locks are battery powered and not connected to a central system; key management is done by software on a disconnected server. VingCard markets primarily to hotels, although the lack of centralization probably limits their use to smaller establishments.

2.1.8 TimeLox

TimeLox markets to the office and hotel segment.[19] Their products use magnetic cards, optionally in tandem with a keypad. Figure 2.8 shows one of these systems installed in a door. Like the VingCard offering, key management is done via a disconnected server.

2.2 Target Market

In the early exploratory phase of the project, a key task was to decide on a target market for our system. The goal was to find a market that was not well served by the existing commercial systems, but was still in need of the functionality provided by an electronic locking system. This would allow an innovative project that would lead to a useful end result, as the system would have few competitors in the commercial marketplace.

Figure 2.5: MaxLock Mounted in Wood

Figure 2.6: Best Basis V Lock

Figure 2.7: Ving Card System

Figure 2.8: TimeLox System installed in door

2.2.1 Possible Markets

There are several target markets to consider. The main ones considered here are systems for the home, universities, and hotels.

Home System

The home system is the simplest of the applications. There are few locks, and all are clustered in one relatively small building. There are few users and user turnover is exceedingly rare. This obviates almost all need for any kind of centralized system. However, there are several cases where new users may be added. These can be divided into two categories, permanent and temporary additions. Permanent additions could be things such as children becoming old enough to have their own keys. Temporary additions would be things such as visitors needing temporary access. Usually, in either case, people given access to a home have a high level of trust with the homeowner, so things such as key removal are not as critical. One thing that is more complex with a home system is the possible need for remote access. If there are access limitations based on time, these may need to be modified remotely if, for example, a cleaning service needs to reschedule to a different day. This is not as important for business or university users, as there is usually an employee on site. Cost for a home system must be very low. Most homeowners are completely satisfied by brass keys, and would not be willing to pay significantly more for the greater flexibility of an electronic lock system.

University Campus

University systems are far larger and more complex than home systems. The distribution of a large number of locks over a wide area makes management a much more complex task. The user turnover in a university environment tends to occur in bursts around the change of school terms. It may be reasonable to have a centralized management system that is only active during these high turnover periods. Key removal mostly occurs on a scheduled basis. There is sometimes uncertainty over when a key expiration will occur, such as for open ended projects. This is not of large importance to key removal, as it is usually not critical to remove access quickly on short notice. Requiring an employee to travel to the physical door to deactivate a key is acceptable for unscheduled key removals due to their rarity. It is probably not acceptable for large adds and removals to require manual operation.

Universities may be willing to make a higher initial monetary investment in exchange for greater flexibility. The focus is on improving level of service while minimizing recurring expenses. Universities have trained personnel overseeing their security systems, so simplicity of operation is not as large a concern as it is for home systems.

Hotel

Hotel systems have several key differences from university systems. User turnover in a hotel is brisk and continuous. Hotels are the case that would benefit most from centralization, especially connected centralization. One thing that simplifies hotel systems is that nearly all key removal can be handled with expiration. Another thing different about the hotel case is that, unless the same card is used for systems outside the hotel, it is exceedingly rare to add access to an additional door for an existing key.

Like universities, recurring cost tends to be more important than the initial start up cost. Hotels may be willing to spend a lot for an electronic system if it greatly reduces the total cost of maintaining the locking systems. The cost of the keys themselves is one of the more significant factors, since hotel patrons are likely to either forget to return their keys after leaving the hotel, or knowingly take them outright. An electronic key that costs far less than a mechanical key would be a significant selling point.

2.3 Chosen Target Market

Our market research indicated that large organizations, such as hotels and university campuses, do not rely heavily on an extremely secure lock/key system. Instead, their security comes from substantial monitoring, including full logging of door accesses via a central system, motion detection, and alarms.[13] In addition, these large organizations have a staff available to respond to alarms and events in logs. This is because adversaries of large organizations usually have

the resources to defeat the physical aspects of security (for example, destroying doors and windows), and so even a very secure lock might be a futile defense.

On the opposite end of the spectrum is the home market. Many homeowners might benefit from the advantages of an electronic system, such as improved security and the elimination of expensive lock replacement if one or more keys are lost. On the other hand, they might not believe that replacing their existing mechanical locks is worth the cost or effort.

As a result, we have chosen to target an intermediate market, which contains small liberal-arts/community colleges, high schools, small apartment buildings, and other smaller organizations. These have a need for something more advanced than mechanical keys, due to the need for semi-frequent key creation, managed access control, and improved security. In addition, these entities will not have a significant staff to handle log monitoring and alarm response. As a result, a good locking system on doors is essential for this market.

2.4 Cryptography

Cryptography is a method of scrambling parts of a message in accordance with a defined procedure or protocol to provide confidentiality, integrity, and authentication. Cryptosystems provide confidentiality by significantly reducing the probability that any party other than the intended recipient of the message will be able to decode the original message. Integrity is provided by increasing the difficulty for a third party to alter a message during transmission. Cryptosystems provide authentication because if a message can be properly decrypted then the probability is high that the sender and receiver are who they say they are.[16] Most cryptosystems work by supplying plaintext, such as written English sentences, and a key, which is an existing value of pre-determined length, to a mathematical function. The result of the function is known as ciphertext, and cannot be decrypted back into plaintext without the appropriate key.[4]

The most basic cryptosystems are based on private or symmetric-key cryptography. One of the first true cryptosystems to be classified as a symmetric-key algorithm was the Vernam cipher developed by Gilbert Vernam in 1918. The Vernam cipher operates on streams of data by performing a bitwise exclusive or (XOR) of the the message with a random cryptographic key. Since XOR is commutative, encryption and decryption using the Vernam cipher are identical. Almost all private, symmetric-key cryptosystems are based in part on the idea developed by Gilbert Vernam.

As of 1998, in the United States, the Advanced Encryption Standard (AES)[1] replaced the Data Encryption Standard (DES) as the primary method of cryptography used for all sensitive data. AES uses a layering system and a variable key length to provide enhanced security and speed over previous ciphers. In addition, its core algorithm is based on modern mathematical structures, called Galois fields, that are relatively new to cryptography but serve as a greater defense against attacks than older cryptographic schemes. The AES algorithm was also designed to be easily implemented on a variety of microprocessors and

custom integrated circuits.[14]

One use for cryptography is authentication via challenge-response. In this scheme, one party, the challenger, will encrypt a message and send it to the second party, the receiver. The receiver will then perform a small previously-agreed-upon operation on the data and re-encrypt and send back the message to the challenger. The challenger will examine the response and verify that the operation was successful, and if so, the challenge-response proves authentication.

One specific attack on cryptosystems is the replay attack. This involves an observer recording the complete encrypted transaction and playing it back to the receiver at a later time[4]. This is useful for an attacker when the general contents of the message is known, but the cryptographic key is unknown. This can be avoided by using a nonce, or semi-random number, as part of the message.

Another common attack used is the brute-force key search. In this case, an attacker will decrypt the ciphertext with every possible key and determine if the resulting plaintext is valid. In most cases, only one key out of all the possibilities will decrypt to legitimate data (for example, English sentences), and thus the attacker can assume he has carried out a proper decryption. However, the number of possible keys is usually so large that this attack requires a vast amount of computing power.

By combining our research of existing systems with our market research, we can produce a set of requirements for the system. These requirements enumerate the features necessary for a system to supply what users want and to compete in the marketplace.

Chapter 3

System Requirements

This chapter lays out the requirements for our locking system. Certain requirements must be met to successfully serve our chosen target market. These requirements fall into two broad categories, the requirements for the individual end user, and the requirements for the overall system. Also discussed are requirements for the cryptography portion of the system.

3.1 End User Requirements

The most important requirement from the end user's perspective is usability. While the security of the system may be hard for the end user to quantify, usability of the system is obvious. For the most part, the end user will not be concerned with the security of the system. The usability however, will affect their day to day lives.

The system should be as easy to use as a mechanical key. If there is a complicated procedure required to use the system, users will attempt to circumvent it. This can lead to compromises in security.

The keys should have durability approaching that of mechanical keys. It may be unreasonable to build electronic keys that withstand as much physical stress as mechanical keys. For example, it is doubtful that an electronic device could withstand as much heat as a brass key. A brass key can easily withstand the heat of a common clothes dryer. It is doubtful that this is possible with an electronic key. The key should be reasonably durable, such that it can easily survive routine handling and use.

The keys users carry need to be small and lightweight. In order for the system to be a compelling alternative to mechanical lock systems, it must not induce physical burdens on the user. If the user needs only one key for all of the doors in a system, this provides an important advantage over mechanical lock systems.

The user should not have to be concerned with power requirements or maintenance. Ideally, the key should be powered by the door, or contain an indefinite

power supply. The user should not need to replace or recharge batteries.

3.2 Overall System Requirements

The costs of the system must be reasonable, and there are several cost areas that must be considered. There is the one-time cost associated with the system. This includes the cost of any centralized management system, and the training of users to operate this system. Next, there is the per-door cost. This includes the cost of the electronic lock mechanism on the door, plus the cost of any required wiring. Finally, there is the per-user cost. Per-user costs include the cost of each key and the cost, if any, of training users how to operate the key.

In order to be of any use, the system should provide at least as much security as a mechanical lock system.

The system should not require much maintenance. It may be reasonable to expect maintenance personnel to change batteries in the doors once a year. If that will be the case, the system should be designed to run for two to three years between battery changes to provide plenty of cushion. It would be most optimal for the doors to require no battery replacement schedule at all. This could be done by charging the internal battery from some external source, such as, a photovoltaic cell, capture of energy from the user opening the door, or connection to the building power system.

To be useful, the system must provide flexibility for key management. The management system must be easy to use and fully featured. This should include methods for database backup and restoration, so that the state of the system may be restored if a mistake is made in operation.

3.3 Cryptography and Security Requirements

When the concept of electronic key cards is introduced into a physical lock system, many issues are raised. With mechanical keys, the only major point of possible failure is at the physical locking mechanism within each door. Electronic key cards can have a multitude of potential failure points, depending on what type of system they are used in.

It is essential that the protocol for communication between electronic door locks, the electronic key cards used to access them, and the Key Management system not be altered during normal operation. To ensure that data is transfered without possible tampering, encryption may be used.

Once a piece of information is encrypted, such as a command for the electronic door lock to process, it is known as ciphertext.In private key encryption, only entities that know the private cryptographic key used to create a ciphertext may decrypt it. It is reasonable to assume that a private key stored within non-volatile memory on an electronic door lock is safe from discovery. If a user with malicious intent tried to obtain a private cryptographic key from a door lock, they would need to physically break open the lock. Any private keys ob-

tained would be useless, as the door the private keys provide access to has been damaged in the process.

Private cryptographic keys used in the system may also be stored on a central key management system, whether this system be a portable device used to program electronic locks, or a single computer system that manages access locally and later transfers information to each lock. Since a Key Management System provides the means to change any access permissions to doors, the system must be protected. If it is compromised, a malicious user could change access permissions with the ease of a system administrator.

Keeping the aforementioned information in mind, the only concern left for an electronic lock system is how information is transmitted from a Key Management System to each door lock. Whether it be through direct connection of a portable device to each lock, via a wired system, a wireless system, or through the use of Smart Cards, the information exchanged must be protected. It is usually possible for a user to intercept information as it passes between the Key Management System and a door, so the information must be encrypted.

Depending on how many entities are involved in the transfer of information, more than one private cryptographic key may be required. For example, in a system using Smart Cards to transfer information from a central Key Management System to each electronic door lock, three classes of cryptographic keys are needed. One cryptographic key, shared between the Key Management System and the door lock, is used to encrypt information passing from the Key Management System to the electronic door lock. That information is further encrypted with a key shared between the Key Management System and the Smart Card. A third and final cryptographic key is used to issue a challenge and response authentication between an electronic door lock and users' Smart Cards that have been given access to that door.

Chapter 4

Possible Methods of System Operation

Now that the requirements of the system have been defined, we must consider possible ways to meet them. This chapter describes the various possible use cases for a door lock system. The chapter then concludes with the chosen methods of operation for the Crypto-SmartLock system.

4.1 Access Methods

There are three possible methods for gaining access to a door. The user may present a physical object, enter a memorized piece of data, or do both. Using only the memorized piece of data is undesirable for security reasons. Users have a tendency to write passwords down if they are hard to remember. If the password is compromised, the user will never know until someone takes advantage of their access to cause damage. If a hard-to-duplicate physical object is required for access to a door, an unauthorized person would need to steal it to gain access. If this were to occur, the user would notice and report the object missing.

Requiring only the key device maintains the convenience and speed of standard physical locks. Requiring a Personal Identification Number (PIN) has the additional security advantage of stolen cards being useless without the PIN. There are two possible ways of using PINs. The PIN can be stored in the door and used as an additional part of the key, or the PIN can be stored in the card. If the PIN is stored in the card, it can be required in order to decrypt the key.

It may be desirable for the system to sleep for a short time after unsuccessful authentications to help prevent brute force attacks. One way this could work is a binary exponential back-off . For example, after the first failure, the lock would sleep for one second. After the second failure, it would sleep for two seconds. The delay would progress to four seconds, then eight seconds, and so on. The back-off time should be capped at some maximum limit, to prevent a malicious user from causing long lock-out periods. There could also be some sort

of alarm after this limit is reached. This could be an audio alarm to immediately alert security personnel, or some indication to the next person to open the door that there was an excessive number of failed authentications. When there was a successful authentication, the sleep time would reset to its initial value.

4.2 Key Management Interface

4.2.1 Key Addition Methods

There are several methods which can be used to give a user access to a door. One group of methods involves the use of a master key. The master key can be presented to the door and validated as a master. Then the owner of the master key indicates to the door that they wish to add access for another user. This could be done by pressing a button on the door or moving the door handle in a certain way. Then the key of the user to be added would be inserted in the door. The door would generate a cryptographic key and exchange it with this new key. After this, the new key would be able to access the door.

It is also possible for the master key method to be used at a PC instead of at the door. The master key would be inserted into a reader attached to a secure PC. The master key user would tell the software they wanted to add access for a new user to a certain door. The PC would then generate a new key pair to be used for this access. The PC would give the master key instructions for the door to add this new key. Then the user would insert their key, and the system would give them the new cryptographic key. The master key would then be inserted in the door and validated as a master to transfer the new cryptographic key to the door. The users card would then be able to open the door with the new cryptographic key.

The system could also operate by using pre-shared keys. When a door is initialized, it would be given a set of cryptographic keys to be used for authentication. These cryptographic keys would also be stored on a secure PC. When access needs to be added for a new user, an operator would authenticate to the PC and then connect the user's key. The PC would transfer one of the cryptographic keys for the selected door onto the user's key, enabling it to open the door.

The system could make use of a specialized device to add access to doors. The administrator would take the device to a door along with the user's key to be added. The adding device would be connected to the door and the administrator's key. After authenticating the key, the device would accept commands to add or remove users. The key to be added would be connected to receive the new cryptographic key.

If there is a central management system which is connected to all of the doors, the user could bring their key to the central system to have the cryptographic key added. The key would then be transferred to the door over the network connection.

It is also possible to have the central system only connected to the doors

during periods of heavy activity. This way the convenience of the central system could be maintained, without as much of the security risk of centralized control.

4.2.2 Key Removal Methods

The possible methods for key removal are similar. The use of a simple master key is complicated by the need to identify which key is to be removed. Obviously it is unacceptable for the system to require the administrator have the key they wish to deny access. If this were the case, there would be no way to deactivate stolen keys. One possibility is to have additional user interface, such as a keypad, on the door for the administrator to enter the ID of the key to be removed.

The key to be removed could also be selected by the use of a special key removal device. This would be the same maintenance device as the one mentioned under key addition.

There are several types of expiration that could be used. The lock could store a time at which a key becomes invalid. When this time is reached, the key is deleted. Another possibility is expiration based on number of uses. The door would store a number of uses remaining and decrement it on each use of a key. When the number reaches zero, the key is deleted. Lastly, keys could be expired due to inactivity. When a key is set up, the lock stores an inactivity time limit for it. Each time a key is used, the door would store the time. If the time since last activation reaches the limit, the key would be deleted.

The viral method of key removal is more of an intellectual curiosity than a useful system. When a key should be removed, an administrator adds a secure message for its removal to someone's key. Whenever the person uses the key, the lock will store this message. Then, the lock will pass the message to all keys it authenticates, spreading it throughout the system. The main problem with this is that there is no indication to the administrators when the disabling of a key is complete.

If an entire door needs to be cleared and reinitialized, this can be done with a mechanical override. This would take the form of a reset button or jumper protected by a mechanical lock mechanism. This mechanism would be accessible only on the inward side of the door. It would be designed to be mechanically secure and conspicuous when accessed by unauhtorized personnel.

4.3 Overall System Paradigm

4.3.1 Connected vs Disconnected

This section describes the difference between a connected and a disconnected system. In a connected system all doors are operating as an ad-hoc network with or without a centralized Key Management System. In a disconnected system, doors operate in a stand alone manner. All programming must be performed on site.

Connected systems typically behave in a more secure manner because the ad-hoc network allows for more advanced monitoring of individual door status. Having a connected system with a centralized Key Management System allows for rapid expansion, high scalability, and will allow administrators to monitor accesses, and system status in real-time without the need to make an on-site visit to the door.

Disconnected systems are usually implemented in smaller settings than connected systems because of the need to perform on-site programming, and the limited memory available in the door. The disconnected system is also typically less secure because monitoring the status of the door and the accesses is not an option.

4.3.2 Centralized vs Decentralized

This section describes the differences between a centralized and a decentralized system. A centralized system is one that uses a central key management system to keep track of and make changes to the state of the system. A decentralized system uses some third-party method to keep track of users and requires on-site programming of the doors.

Centralized systems bring more convenience to the system as a whole because there is one central location where all users and doors can be managed, and where any door can be programmed.

Decentralized systems are often used where the system is small enough such that keeping track of accesses is manageable without the use of a computer, and where users, doors, and access permissions rarely change.

4.3.3 A Disconnected, Centralized System

The interface of the system we will implement is a disconnected centralized system. There will be a central computer for key management, but it will not be directly connected to the individual door locks. This provides the advantages of centralized key management without the cost and security issues of connecting all of the doors to the central computer.

When doors are installed, a cryptographic key will be shared with the central computer system. After that, this cryptographic key will be used to encrypt and authenticate all communications from the central system to the door. All communication between the key management system and the doors will occur via the keys. This is a flexible new approach that we do not believe has been used in any existing lock system.

4.4 Chosen Method of Operation

The system we will design and implement will be composed of a disconnected centralized key management system, door locks, and physical electronic keys. The keys will be granted access to the locks by the key management system. In

addition, the key management system will communicate to the locks by issuing commands, and the physical keys will serve as the communication medium for those commands. The locks will also communicate with the key management system by confirming the commands it has executed to ensure that the key management system always has up-to-date information. All of these communications will be encrypted. Figure 4.1 shows the three types of elements in this system and the communication paths between them.

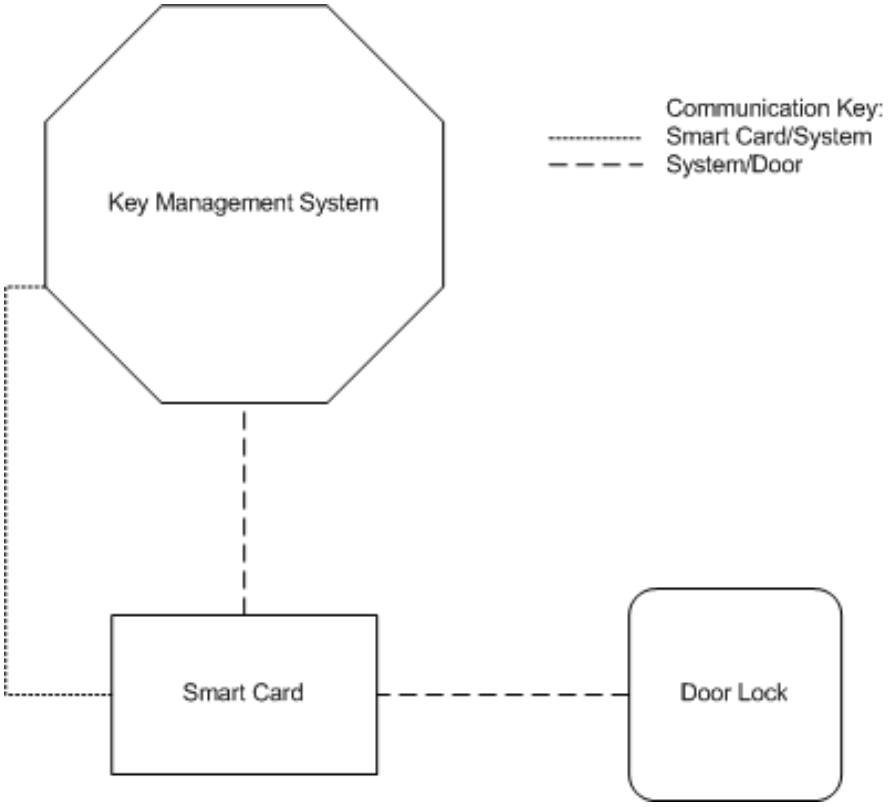

Figure 4.1: Overall System Block Diagram

Chapter 5

Design Alternatives

Now that we have considered the requirements and possible methods of operation, this chapter will consider the design alternatives for implementing the system. There are several key components that must be selected. Physical interfaces will be considered, followed by consideration of possible products which implement this interface. We will also analyze several possible microcontrollers and select one for our implementation.

5.1 Physical Interfaces

Door locks can interface with keys in a multitude of ways. Most people are familiar with mechanical locking systems that utilize a machined key. Electronic lock systems have many more possibilities than this, but also many more considerations.

5.1.1 Criteria

The important criteria for a physical interface is that it meets the end user requirements detailed earlier. Specifically, it must be easy and convenient to use.

5.1.2 Possibilities

Physical Contacts

There are many interface possibilities for a physical security system. First is physical contacts. A door system has a reader with electrical contacts, and the user presents a key with matching contacts. This may consist of, for example, a credit-card sized plastic key that is inserted into a door. Electrical contact is made, and information exchanged to authenticate and validate the user.

Keys with physical contacts exchange data via an external bus, and unauthorized users may attempt to monitor information that is exchanged in order

to break into a system. It is therefore important that the information exchanged between a key and host system not compromise system security if intercepted. Also an important concern is durability of physical interfaces. Metal contacts eventually wear down after prolonged usage. Keys and host interfaces may need to be replaced over set time-frames to ensure correct functionality.

Wireless: Infrared and Radio Frequency

Another interface paradigm consists of wireless devices. Two specific possibilities include Infrared (IR) and Radio Frequency (RF) communications. Infrared is line-of-sight based, which makes it difficult for unauthorized users to examine wireless packets of information. The LEDs that produce the infrared light are low power. Reliability and speed may be lacking, though. Radio frequency lends much more flexibility and power, but introduces other concerns. There are many standards for data exchange via radio frequencies, such as WiFi and Bluetooth. Range of operation may be small or vast, and is omni-directional. RF interfaces tend to draw more power than IR in general, though some key systems actually transmit power to the key from the door via RF in close proximity.

Infrared or radio frequency packets may be intercepted during communication with a host system. Like with physical contact systems, it is important that the information exchanged will not compromise system security. Durability is clearly superior to physical contact systems, since there is no friction or wear from use. Wireless systems, however, are susceptible to interference, possibly causing undesired operation or compromising security.

Challenge-Response Systems

Lastly are challenge response systems based on output data from a host, and corresponding unique input data from a user. In addition, the user has a key system that facilitates decrypting output data from the host. Typically these consist of a screen to display visual data from a lock system, and a keypad to accept input from a user. For example, the host may present a specific alphanumeric string of ciphertext to a user. The user inputs this ciphertext into their key, perhaps through the use of another keypad and screen. The key will modify the ciphertext in a specific way that the host can validate, and display another ciphertext to the user. This is entered into the host system via a keypad, and the user is accordingly allowed or denied access. This same challenge-response concept is often used in the aforementioned interface types internally, making the exchange transparent to the user.

What You Have and What You Know

As a supplement to any of the above interface types, a unique piece of information may be added as a requirement for authentication of a user. This is specifically something fairly small that may be remembered by an individual, and used in addition to a physical key. An implementation of this concept that

may be familiar to many is the use of a PIN, or personal identification number, at automated teller machines. A user inserts their ATM or debit card into the machine, and is prompted for their PIN before they are allowed to engage a transaction. Similarly, a door lock system may require a physical key to be inserted, and then a pin number or alphanumeric phrase entered via a keypad before access is granted. A system may also work by facilitating input of the PIN on the key itself instead of the lock system. This concept negates maintaining the convenience and speed of a standard physical lock system, but has additional security advantages. If a key is lost and found by an unauthorized person, they may not gain access unless they know the PIN or passphrase.

5.2 Smart Cards

One specific implementation of a key card utilizing physical contacts is the Smart Card. These keys share the same physical dimensions with standard plastic credit cards, and may be easily stored in a wallet or worn on a person's exterior through the use of a lanyard or clip. They may have printed information on their surface, such as a photo ID or other useful information. Information is exchanged with a host system through the use of data, clock, and power pins on the exterior of the card. There are also versions that utilize an RF interface instead of electrical contacts.

A Smart Card is not simply a carrier of keystring data, it contains a complex embedded circuit. This circuit consists of a Central Processing Unit, Read-Only Memory, Random-Access Memory, and non-volatile memory. The CPU has at least the power of an 8-bit microcontroller such as the Intel 8051 or Motorola 6805. Many Smart Cards far exceed this. Some add a dedicated cryptography co-processor to facilitate efficient and timely encryption and decryption with complex algorithms. The ROM is typically between 8 kilobytes and 32 kilobytes. It contains an Operating System specific to each type of Smart Card. The RAM allows for buffering information during data transfer. Lastly, the non-volatile memory is used as a key space and miscellaneous storage specific to a Smart Card system implementation.

Data integrity is the central concept Smart Cards were designed for. The host system and Smart Card agree about changing the state of data on the card and server. Smart Cards are very flexible in their possible usage, though, so they easily fill our requirements for a physical security system, in which most operations do not modify existing data. For key management systems, though, data integrity will play a very important role.

Communications between the Smart Card and host are encrypted. Also, the host and Smart Card authenticate each other before starting a transaction. Many Smart Cards make it possible to implement challenge and response systems between a host and the card, which is what our project will utilize.

The non-volatile portion of memory on the card stores dynamic information in a specific filesystem. Most card operating systems support the ISO-7816 Smart Card standard for file storage. This consists of a singly rooted, directory-

based hierarchical file system, long alphanumeric names, and a permissions system based on identities. The identities concept only allows authorized hosts to access any given piece of data on the card, according to pre-determined information.

Considering the flexibility and power of Smart Cards, they are relatively inexpensive as well. Combined with the ability to easily implement cryptography algorithms and challenge-response systems, Smart Cards satisfy all requirements of this project. Many types and varieties are readily available, as are development kits and application examples.

5.2.1 Proprietary Smart Cards

The most abundant type of Smart Card available on the market is the proprietary Smart Card. This usually means that a company started with a blank Smart Card, and designed an operating system and file system to run on it. Many times, the physical Smart Cards are accompanied by a suite of development tools and documentation. Most of these systems, though, are designed to cater to a small, specific market segment. Due to this fact, finding a proprietary Smart Card that satisfies the Crypto-SmartLock project requirements would be difficult.

One example of a proprietary Smart Card is the Advanced Card Systems Ltd. (ACS) model ACOS1 Smart Card. Five of these cards came bundled in a Smart Card development kit that was purchased for the Crypto-SmartLock project. The ACOS1 Operating System design only makes it possible to store one Terminal Key at a time. This key makes it possible to securely authenticate the terminal the card is communicating with. Our multiple-door system requires many unique terminal keys. The ACOS1 Operating System was designed with specific target implementations in mind. Its capabilities are limited in several key areas that are important to this project. The most important limitation being the set limit of four to five cryptographic keys that may be utilized to authenticate a host system. In addition, these few keys were intended for specific purposes, and may not have all been applicable to a customized design. The Operating system's internal set of commands are limited to specific applications, and may not satisfy requirements for this project. The ACOS1 Operating System is not unique in this concept, as most Smart Cards and corresponding development kits on the market come with Operating Systems designed for specific situations that greatly differ from the goals of this project. Due to this general limitation, other types of Smart Card are more suitable.

5.2.2 "Blank" Smart Cards and Designing an OS From Scratch

Few and far between are "Blank" Smart Cards. These cards do not come with a specific OS implementation, but instead allow a completely custom implementation to be designed from the ground up. For development, these cards replace the ROM section with another non-volatile memory chip, such as flash ROM or

EEPROM. The developer designs and loads an OS in this space, then configures the other sections like a standard Smart Card.

Developing a Smart Card system from the ground up is a serious undertaking, and such flexibility is not necessarily required for this project. This route also requires a suite of potentially expensive software, such as a chip compiler to produce the correct byte-code for the Smart Card CPU, and an SDK to manage Operating System design and uploading to the card. Also, availability of blank cards is lacking. Most vendors do not make mention of such cards, and only sell pre-made implementations.

5.2.3 BasicCards

Fortunately there is something in between using a completely off-the-shelf Smart Card system and implementing a Smart Card operating system from scratch. The BasicCard from ZeitControl Cardsystems GmbH is user programmable using a variant of the Basic language. These cards include an operating system in ROM, but also allow the user to add custom commands in the EEPROM. The included operating system provides an implementation of the ISO/IEC 7816-3 T=1 transfer protocol and a filesystem. Also included are implementations of various cryptographic algorithms depending on the card version. The remaining algorithms are included as optional software libraries.

The BasicCard provides a good balance of flexibility and ease of use. Proprietary Smart Cards do not require programming, but if the card vendor did not anticipate an application, they cannot be extended. Blank Cards provide the ultimate flexibility, but the costs, in both software tools and programming required is prohibitive in anything but the highest volume applications.

The BasicCard provides full programmability without the necessity of designing a full operating system from scratch. Because all of the standard Smart Card functionality is pre-implemented, system implementers need only write the CardBasic code specific to their application.

Table 5.1: ZeitControl BasicCard Enhanced - Currently Available Cards

Version	RAM	EEPROM	Encryption
ZC3.7	256 bytes	2 Kbytes	DES
ZC3.9	256 bytes	8 Kbytes	DES

ZeitControl BasicCards are broken down into three families. There are Compact BasicCards, Enhanced BasicCards, and Professional BasicCards. The Compact BasicCard family does not include a file system or encryption support, so it did not satisfy our requirements for this system. The Enhanced Family comes with file system support, floating point support, built-in DES cryptography, and varying EEPROM sizes. The two types of Enhanced BasicCards that were available for purchase on the ZeitControl website are listed in Table 5.1.

Professional BasicCards provide a great means of possible expansion for this

system, as they include much larger EEPROMs and include hardware for AES cryptography acceleration. There are also two types of Professional BasicCards available for purchase, and these are listed in Table 5.2.

One requirement for the Crypto-SmartLock system is that the encryption used by each element in the system must be powerful and secure. Unfortunately, the Enhanced BasicCards support DES encryption by default, which is not powerful enough for this application. Luckily, they support adding AES cryptography by adding software to the card. The only downfall to this is the loss of about two kilobytes of EEPROM memory to store the AES encryption and decryption library.

The Professional BasicCards go beyond satisfying requirements for a Crypto-SmartLock, but cost twice as much as the Enhanced BasicCards. For the purposes of the Crypto-SmartLock's target market, we chose the ZC3.9 Enhanced BasicCard.

5.3 Microcontrollers

The most important element of modern embedded systems is the core microprocessor and its immediate peripherals. As a result, a wide range of microcontrollers are available that integrate a multipurpose processor with the most commonly used secondary components, such as multipurpose digital input/output, and short- and long-term storage. An important side effect of including all of this functionality on one chip is a much lower cost than a microprocessor system, making a microcontroller an ideal choice for our door lock.

5.3.1 Criteria

The important qualities of a microcontroller for a system of this type are power usage, processing speed, and cost. Because the system will be expected to operate for long periods without changing batteries, very low power consumption is critical. The processor must be powerful enough to perform cryptographic operations at a reasonable speed. The microcontroller should be relatively inexpensive to keep the overall cost of the system down.

Table 5.2: ZeitControl BasicCard Professional - Currently Available Cards

Version	RAM	EEPROM	Encryption
ZC5.4	1 Kbyte	16 Kbytes	AES, DES, Elliptic Curve
ZC5.5	1 Kbyte	31 Kbytes	AES, DES, Elliptic Curve

5.3.2 Possibilities

In order to use modern security techniques, a good microprocessor or microcontroller should be at the core of an electronic lock system. Because this is a specialized embedded system, a microcontroller will provide all the features necessary for implementation in a single package, reducing cost. We limited our search to low-cost 16-bit microcontrollers, which are better equipped to handle mathematics-intensive cryptography, but should not have features we would not use in our system.

Texas Instruments MSP430

The Texas Instruments MSP430 Microcontroller is advertised as being specifically for low-power applications.[8] Its current draw while active is stated as 250 μA per megahertz, which will makes it ideal for a battery-driven system. The 430 also has a hardware multiply subsystem, which could be useful in cryptographic operations. Different versions of the 430 are available with 14 to 48 I/O lines, and can operate at up to 8 MHz. The chips start at $0.49, and development tools start as low as $20.

Rabbit Semiconductor 2000

The Rabbit is a full-featured microcontroller, originally developed for use in network embedded systems.[15] In low-power mode, the Rabbit only consumes $100\mu A$ of current. The chip can operate at up to 30 MHz, and includes 40 I/O lines, and claims to be optimized for mathematical operations (such as those necessary for cryptography), including a hardware-based multiply component. It also has a 48-bit internal clock, useful for calendar-oriented and time-sensitive applications. Unfortunately, the Rabbit 2000 starts at $8.75, which is much more expensive than all but the most high-end microcontrollers on the market. Development tools start at $139.

Atmel ATmega

Atmel Corporation is a large and well-known manufacturer of microcontrollers. Its ATmega line includes 53 I/O lines and runs at up to 16 MHz (which translates to 16 MIPS).[3] Power consumption is rather high at 2.5 mA, as is the pricing, which starts at $5.05. Development tools are available for as low as $79.

Motorola HCS12

Motorola is another major player in the semiconductor market. Its HCS12 line of microcontrollers seems to be their midrange offering, with performance at up to 25 MHz and 53 to 91 I/O lines.[12] However, the chips are priced starting at $19.26, and power consumption is 2 mA/MHz. More importantly, development tools start at $1,500, making this option unattractive.

Arizona Microchip PIC16

Although not a 16-bit microcontroller, we looked at the PIC because WPI already owns the necessary development tools and we have experience with them. These chips have 22 to 38 I/O lines and operate at up to 20 MHz.[11] Power consumption is less than 2 mA at maximum, and can be as low as $10\mu A$ at very low clock speeds (32 KHz). Pricing starts at $1.17 for the microcontroller and $200 for development tools, although, again, development tools would probably not have to be purchased.

5.3.3 MSP430

We decided on the TI 430 platform for our project. This microcontroller line is most attractive due to the low cost of the chips and development tools relative to the features included. The power consumption also provides for flexibility in the power source we choose. Another key feature is the fact the the 430 flash memory is in-circuit programmable. This greatly simplifies the design and construction of the door lock system, because there is no need to interface with external non-volatile storage devices. Another selling point of the MSP430 platform was the fact that one of the project advisors had worked with the platform before and was confident it would meet our needs.

5.3.4 MSP430 Development Systems

We considered several possible development boards for the prototype door system. These candidates are detailed in Figure 5.3. We decided that the best development board choice is the SparkFun 449-STK. This board provides the necessary features at the lowest cost. The SoftBaugh board does provide more features, but these are not necessary. The one feature advantage that might have been worthwhile is the two megabit serial flash. However, the 60 kilobytes of flash internal to the MSP430F449 should be sufficient for our prototype, and an external Flash or EEPROM chip can be added to the SparkFun board.

Table 5.3: Development Board Comparison

Vendor	**SoftBaugh**	**TI**	**SparkFun**	**Sparkfun**
System	ES449	TI F41x FET	449-STK	1121-STK
Cost	$199	$99	$65.95	$19.95
Micro	MSP430F449	MSP430F413	MSP430F449	MSP430F1121
Compiler	none	IAR 2k limit	none	none
Interface	none	FET	none	none
Serial port	yes	no	yes	yes
Display	yes	no	yes	no
Additional IO	all ports	pin headers	32 (digital)	14 (digital)
Debugging support	JTAG	JTAG	JTAG	JTAG

Chapter 6

System Design and Specifications

Now that we have explored the system requirements and the design alternatives that could be used to meet them, we shall describe the design of our prototype system. The communication protocol will be described first. The protocol is the backbone of the system. Without the carefully designed protocol, the system would not provide good security and usability to meet the requirements.

The protocol design leads into the design of the system to implement the protocol. The central system design and implementation are described in Sections 6.2 and 6.2. Then the key card design and implementation are described in Sections 6.4 and 6.5. The door system design and implementation are described in Sections 6.6 and 6.7. Section 6.8 describes the flow of operation of the system. Finally, Section 6.9 describes a series of possible attacks and how they are prevented.

6.1 Communications Protocol

A well-designed communications protocol is essential for the security of the overall system. In addition to being secure in their own right, the different components must be able to relay information amongst themselves regarding access control. This section describes the protocol we have designed to meet these requirements.

6.1.1 Communication Elements

Our physical security system is made up of door locks, Smart Cards for access and relaying commands (discussed below), and Key Management System software running on a standard personal computer to manage access. Each element in the system must be uniquely identified. As a result, each card has a unique identification number (card ID) ranging from 0 to 1023. Likewise, each door has

an identification number (door ID) ranging from 0 to 255. Uninitialized doors use a default ID of 255, and ID 0 is reserved for the Key Management System. Uninitialized cards use the ID 65535, which does not conflict with the standard card ID range of 0 to 1023.

6.1.2 Cryptographic Keys

There are three types of cryptographic keys used in the overall system, all of which are 128-bit AES keys. (See Section 2.4). The most numerous is the authorization key (K_{Auth}). This is a key shared between each door and each card that has access to it. Each card may store as many as 254 of these keys, and each door can handle up to 1024 of them. Every card that has access to a door has a unique K_{Auth} for that door; any given K_{Auth} is known by only two elements in the system (one card and one door). Because each K_{Auth} is unique, if one card is compromised, its keys can be discarded in favor of new ones. Any 128-bit value can be a K_{Auth}, except for the value $2^{128} - 1$, which is reserved as a "sentinel key" on the door memory, to indicate the absence of a key for a given user.

The second type of cryptographic key is called K_{SD} (key system-door) and is used by the key managment system to encrypt commands destined for a door, along with confirmations from a door to the key management system. The key management system knows every K_{SD}, but each door only knows the one it uses to decrypt its own commands. Cards do not know any K_{SD}, so that the contents of a command can never be known when the command is in transit. There could be as many as 254 of these keys in use in a system.

The final type of cryptographic key is used for communication between a card and the key management system, and is called K_{CS} (key card-system). As with K_{SD}, the key management system knows all of these keys, and each card knows only one. When the key management system has a command to be relayed by a card, it has some information about the command that only the card must know. This information is encrypted with K_{CS}.

6.1.3 Commands

Because there is no direct connection between the key management system and the individual door locks, the cards themselves serve as their communication medium. When the key management system issues a command to a door (e.g. "add access for this person"), it does so in the form of a command. A command is made up of several elements, the first of which is the opcode. This indicates the type of command; the possible commands are ADD USER, REMOVE USER, SETUP DOOR, CHANGE DOOR KEY, and NO OPERATION. A command may also have a body of data included within it. For example, an ADD USER command will have to include the card ID and the K_{Auth} for that card.

Every command issued is also labeled with a unique transaction ID, which serves to prevent "replay attacks" (see section 2.4). The key management system maintains the current transaction ID for each door, and doors must keep

track of which transaction IDs have already been attached to commands. Thus, if a command arrives with a previously-used transaction ID, it is known to be invalid. A door can keep track of 65536 transaction IDs; once this many commands have been carried out, the door should be issued a new K_{SD} and the transaction ID count will start over. Under this scheme, if a K_{SD} is compromised, it will only be useful to a malicious user for a relatively short period of time.

The other two protective measures in a command are the destination door ID, and the nonce. The door ID is included so that the door knows that the command has decrypted to a valid set of data. The nonce, by definition, is a randomly-generated value used only once.[4] When the key management system supplies a command to a card, the card cannot decipher the contents of the command; however, the key management system does give the card the nonce stored in the command. The nonce is used by a door to prove to a card that it is in fact the door it claims to be; it will send the card the nonce it just decrypted to prove this.

Command Header

Every command is associated with a header, which is simply information about a command given to a card. Unlike the command itself, the command header is readable by a card; in fact, the card requires this information for a command to reach its destination. The header contains two nonces; one is the nonce described above, which used by the card to verify a door's identity. The other is used in a similar manner as that nonce, but is actually used by the key management system to verify the card's identity. To avoid confusion, these are referred to as the command nonce and the header nonce. The command nonce is retained by the card until it reaches a door, whereas the header nonce is used and discarded while it is still connected to the key management system. The command header also contains the card's own identification number, to confirm that it decrypted properly, and the destination door ID, so that the card knows where the command should be delivered.

Confirmations and Confirmation Header

When a command is executed, the door immediately generates a confirmation. This is a message to the key management system indicating that it has successfully executed. A confirmation takes the same form as a command, since the key management system will need to know the original command's transaction ID and door ID. The confirmation also contains a nonce, used in the same manner as the nonce in a command, and a special opcode that identifies it as a confirmation and not a regular command. In addition, a door will send a confirmation header in the same manner as it would a command header. The confirmation header contains a confirmation nonce and header nonce, along with the door ID.

6.1.4 System Component Interaction

The activity between two system components (e.g. a card and a door) follows the processes detailed below.

Door-Card Interaction

Figure 6.1 shows the sequence of events that occur when a card arrives to communicate with a door. A rectangle indicates a communication is taking place, whereas a hexagon indicates an operation occurring on one of the two components. Encrypted communication is labeled with the encryption key the transaction uses.

Card-System Interaction

Figure 6.2 shows the sequence of events that occur when a card arrives to communicate with the central system. A rectangle indicates a communication is taking place, whereas a hexagon indicates an operation occurring on one of the two components. Encrypted communication is labeled with the encryption key the transaction uses.

6.1.5 Communication Internals

With any protocol, communication must meet the specifications exactly. Table 6.1 shows the details of the major communication elements in our protocol down to the byte level.

Command Details

Table 6.1: Byte Structure of Door Commands

Index	NOOP	ADD USER	REMOVE USER	SETUP USER	CHANGE DOOR
0	00h	01h	02h	03h	04h
1	Nonce (h)	Nonce (h)	Nonce (h)	Nonce (h)	Nonce (h)
2	Nonce (l)	Nonce (l)	Nonce (l)	Nonce (l)	Nonce (l)
3	Trans. ID (h)	Trans. ID (h)	Trans. ID (h)	Trans. ID (h)	Trans. ID (h)
4	Trans. ID (l)	Trans. ID (l)	Trans. ID (l)	Trans. ID (l)	Trans. ID (l)
5	Door ID	Door ID	Door ID	Door ID	Door ID
6	User ID (h)	User ID (h)	User ID (h)	Door ID new	
7	User ID (l)	User ID (l)	User ID (l)		
16-31		K_{Auth}		K_{Auth}	K_{Auth}

A door command is 32 bytes of ciphertext, encrypted with K_{SD}. The possible commands are listed in Table 6.1.

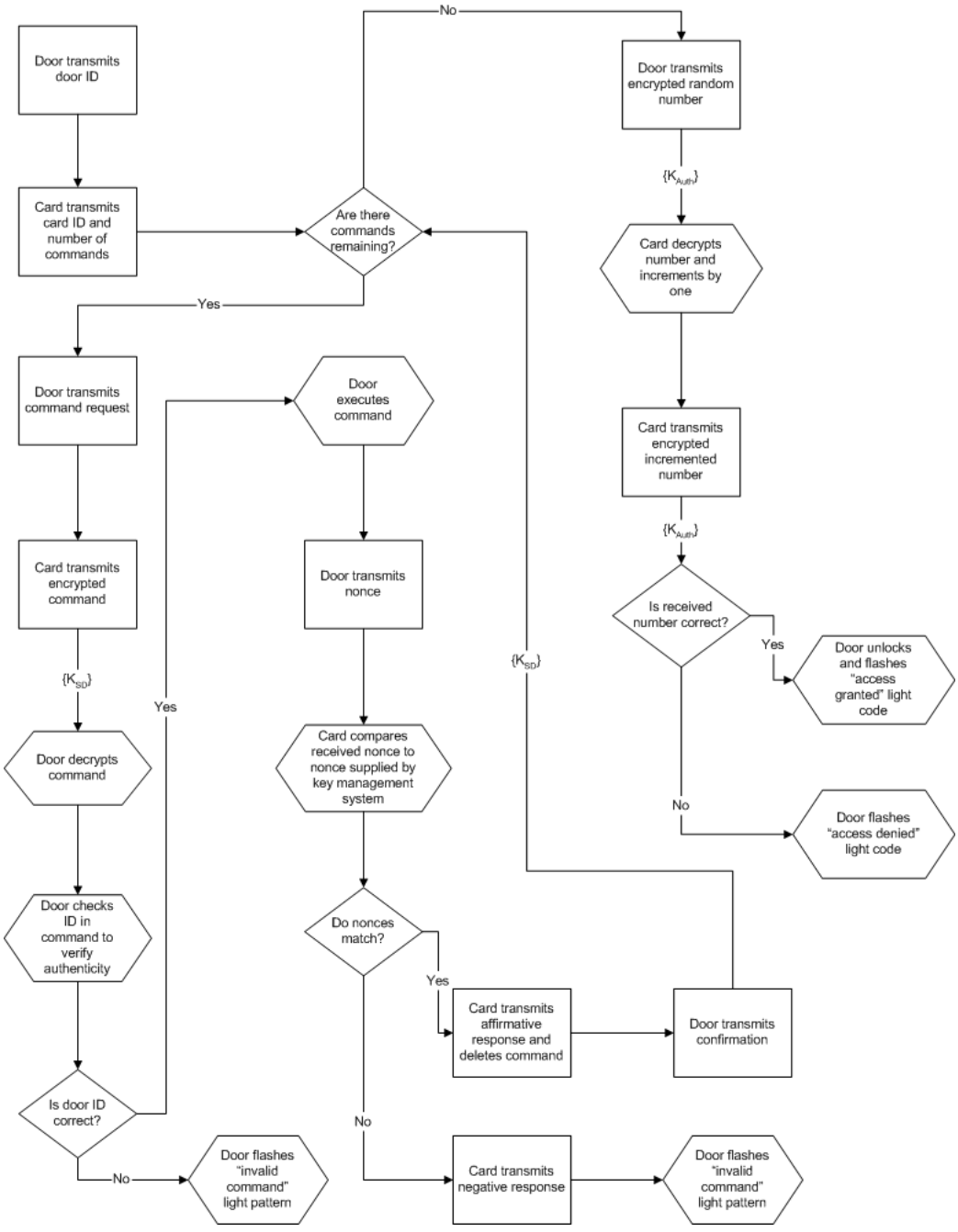

Figure 6.1: Door-Card Interaction Diagram

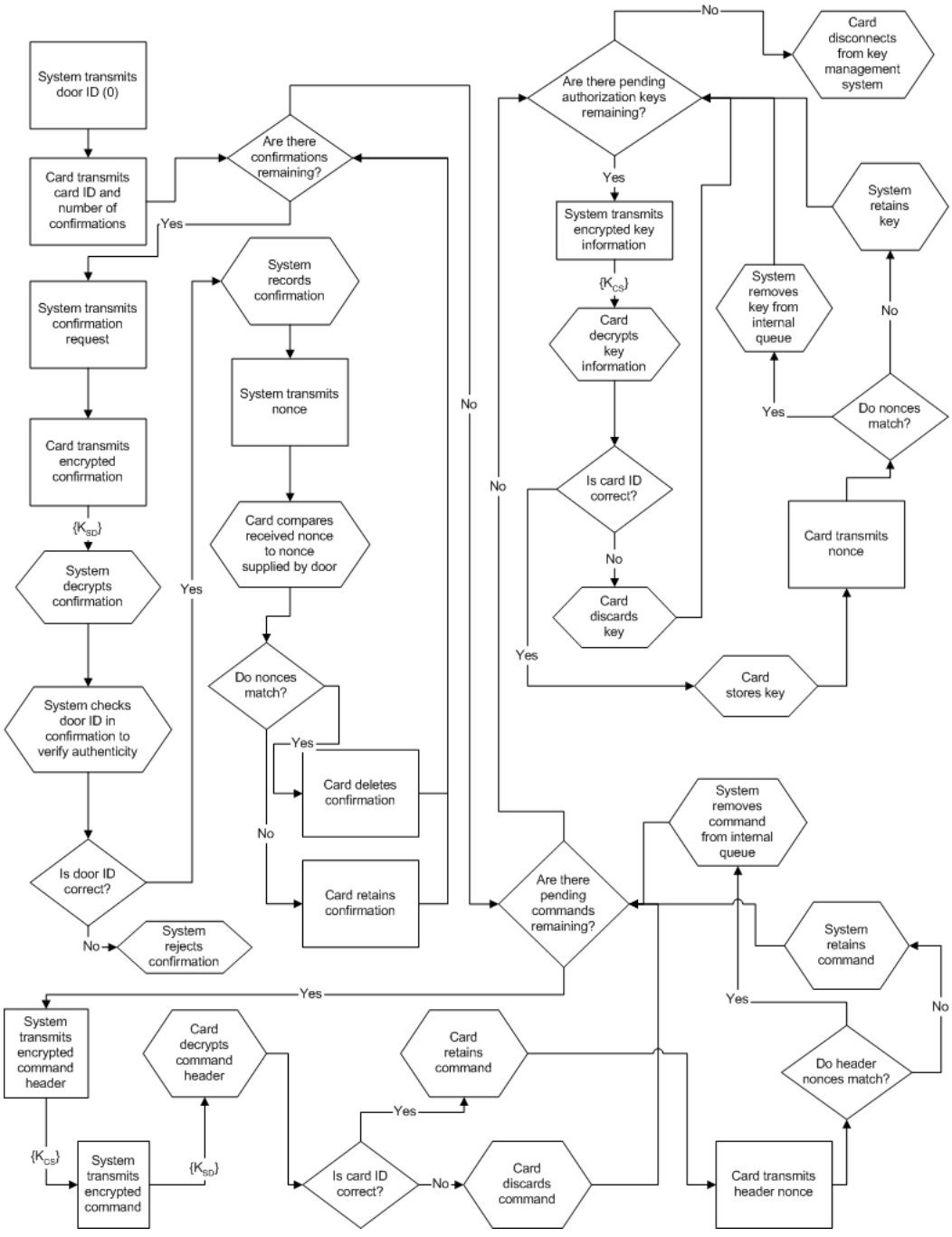

Figure 6.2: Card-System Interaction Diagram

Command Header Details

Table 6.2: Byte Structure of Command Headers

Byte	Contents
0	Header Nonce (h)
1	Header Nonce (l)
2	Command Nonce (h)
3	Command Nonce (l)
4	Destination Door ID
5	Card ID

A command header is 16 bytes of ciphertext, encrypted with K_{CS}. Its structure is shown in Table 6.2.

Confirmation Details

Table 6.3: Byte Structure of Confirmations

Byte	Contents
0	0x06 (placeholder opcode)
1	Nonce (l)
2	Nonce (h)
3	Transaction ID (l)
4	Transaction ID (h)
5	Destination Door ID

A confirmation is 32 bytes of ciphertext, encrypted with K_{SD}. Its structure is shown in Table 6.3.

Confirmation Header Details

Table 6.4: Byte Structure of Confirmation Headers

Byte	Contents
0	Header Nonce (h)
1	Header Nonce (l)
2	Confirmation Nonce (h)
3	Confirmation Nonce (l)
4	Source Door ID

A confirmation header is 16 bytes of plaintext. Its structure is shown in Table 6.4.

Authorization Key Information Details

Table 6.5: Byte Structure of Authorization Key Information

Byte	Contents
0	0x07 (placeholder opcode)
1	Nonce (h)
2	Nonce (l)
3	Door ID
3	Card ID (h)
4	Card ID (l)
16-31	New K_{Auth}

Authorization key information is 32 bytes of ciphertext, encrypted with K_{CS}. Its structure is shown in Table 6.5.

6.1.6 Protocol Rationale

Secure communication between the devices in the system is essential to its operation. Using Smart Cards as the communication medium is an approach uncommon to the field of physical security, and lends itself to vulnerabilities due to the potential for theft and fake cards. As a result, the protocol used for the communication must be properly designed to prevent any attack a malicious user might attempt. We have examined a variety of attacks that could be made on the components of the system, and added functionality in the protocol to prevent them. The following is a description of why our final protocol is set up the way it is, and how it provides security.

Door to Smart Card

The purpose of communication between the door and a Smart Card is to authenticate the holder of the card for access to that door, and relay information to and from the key management system. In the case of authentication, the card must prove to the door that it has the appropriate access. A malicious user must not be able to use a card to pose as a different card, or gain any useful information from observing the traffic flow between a real door, or a false door he/she might create to ease data collection. Our solution is a challenge-response system that uses AES encryption (See Section 2.4). Each door-key pair has a unique cryptographic key that the two devices share. First, the door sends an encrypted random number to the card. Next, this number is decrypted, incremented by one, re-encrypted, and returned to the door.To an observer, the data in transit will not be of any use. A brute-force attack (see Section 2.4) can be used to find the key, as the relation between the plaintext is known, but even the most powerful of today's computers will take years to perform such an attack on AES. By this time, the cryptographic key will likely have changed.

A Smart Card relays information to a door from the key management system in the form of commands, and information in the other direction in the form of command confirmations. In both cases, the card does not know the contents of the information it carries, as it is encrypted with a key shared between a single door and the key management system. This is done so that a malicious user cannot extract or insert information being relayed in a usable form. However, a malicious person could create a false card that supplies random data to a door in order to cause a malfunction, or create a false door that accepts a command from a card to prevent the true destination of a command from receiving it. To prevent the case of a false door, the door must prove to the card it does in fact possess the ability to decrypt the command. In order to do this, a Smart Card is given a random number (nonce) with the encrypted command. This nonce is also present in the encrypted data itself. When a card sends a command to a door, it expects the door to return the nonce, which it can then compare to the nonce it was given by the Key Management System. If they match, then the card knows that the door possesses the proper encryption key and is the door it claims to be. Otherwise, the card should retain the command and not accept a command confirmation from the door. A malicious user could attempt to supply a door with false ciphertext in the hopes that it decrypts to a valid command, but the door checks to confirm that its own door number is present in the ciphertext. Both the number and the format of the command would have to match, so the number of valid commands that could be generated randomly is extremely low relative to invalid ones. This attack is thus impractical.

An eavesdropper might try to monitor communication between a card and a door and record a command, and then create a false card that sends that command again at a later time. However, the doors keep track of the transaction ID attached to each command executed, and this replay attack would not succeed (see Section 2.4 for more information on replay attacks). Although there is a finite set of transaction IDs, so that storage on the Smart Cards and doors is conserved, a roll-around of the numbers should coincide with a reset of the shared key between the door and the system. Any outstanding commands on the cards will become invalid, and the key management system can re-issue them using the new shared key, if necessary.

System to Smart Card

As seen above, the likelihood of a false door successfully delivering information to a Smart Card is very low. As a result, when a valid card returns to the key management system, its integrity is assured. Nonetheless, even if a card did contain an erroneous command confirmation, it would not be encrypted properly, and the key management system would alert the locksmith or system administrator to an anomaly. Likewise, since the Key Management System can be assumed to be physically secure (in a safe, for example), an attacker could not supply it with a false card. Even if this were a possibility, the Key Management System would still reject such a card, since the attacker would not know the cryptographic keys to make its data valid. No matter what the case,

Table 6.6: System Requirements for Key Management System

Parameter	Minimum	Recommended
Operating System	Microsoft Windows®2000	Microsoft Windows®XP
CPU	Pentium II	Pentium IV
Ram	100MB	512MB
Disk Space	4Mb	32Mb
Ports	RS-232 Serial	USB
Reader	PC/SC Compliant	PC/SC Compliant

cryptography will ensure the key management system's integrity.

A resourceful malicious user might create a false key management system, and attempt to use it to modify a valid Smart Card. However, the same mechanisms that prevent false doors also prevent false Key Management Systems. A card will not mistakenly give its command confirmations to a false system, because the system must verify the nonce in the same manner a door would. If the nonce it sends back to the card does not match the nonce the system gave it in the first place, the card knows it is not the real Key Management System. Likewise, any commands the system supplies to the card (either door commands to be relayed, or card commands such as re-initialization) must be encrypted properly; if they do not decrypt to valid plaintext, the system is again revealed to be false.

6.2 Key Management System Hardware

The minimum recommended hardware configuration for the Key Management System is available in Table 6.6. For security concerns it is highly recommended that the PC running the Key Management System not be connected to any network, and be kept in a secure location.

6.3 Key Management System Software

The Key Management System software is the heart of the Crypto-SmartLock system. The administrator adds records of every physical lock to be installed in the system, and every person who will use the system. After that, users are assigned access permissions to doors.

Once users, doors, and permissions are initialized, settings are sent to each door via BasicCards, in the form of commands. The Key Management System allows an administrator to manage commands, and determine which BasicCards will carry a command to a door.

6.3.1 Human Interface Design

The Crypto-SmartLock Key Management System is implemented with a Graphical User Interface (GUI) to allow system administrators easy access to change settings. This GUI was designed as a Microsoft Visual Studio®.NET Visual C++ Windows Forms Application. The end result is a program that follows the Microsoft Windows application paradigm.

All Key Management System functionality is accomplished on six tabs within the GUI, entitled: *Manage Users*, *Manage Doors*, *Manage Access*, *Reissue Commands*, *Assign Commands*, and *Card Reader*. The state of the system is tracked in real time, and noted in a status bar at the bottom of the Key Management System window. The status bar will also display messages to the user about interactions with the system, such as simple instructions or feedback from events. The system state may be saved at any time by using the *File* menu, and selecting *Save Changes*. If a mistake is made, it may be undone in most circumstances by selecting *File*, and then *Revert Changes*. Every time the Key Management System synchronizes with a card, though, the state of the system will automatically be saved, and changes may not be reverted.

Manage Users

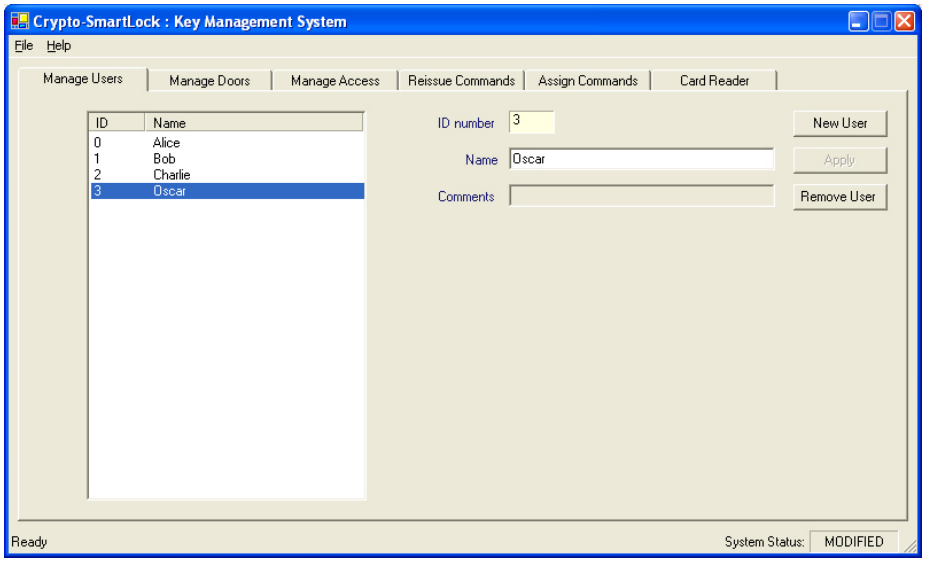

Figure 6.3: Manage Users Tab Screenshot of the GUI

Whenever an administrator needs to add a new user to the system, they may do so via to the *Manage Users* tab, as seen in Figure 6.3. Each user is kept track of internally using a unique User ID that is automatically assigned.

Administrators may specify the person's name associated with each of these IDs. Names can be alphanumeric, and may be changed at any time without affecting the functionality of the system. It may be advantageous for administrators to use a unique name with each User ID so as not to confuse users, but it is not necessary.

If an administrator ever needs to remove a user from the system, they may do so using the *Manage Users* tab. All they need to do is select a user, and click the *Remove User* button. Once this is done, the system will automatically issue commands to remove the user's access to every door in the system. By default, these commands are not assigned to any user's card, so the administrator must use the *Assign Commands* tab to assign them to an appropriate user - most likely an associate of the company.

Manage Doors

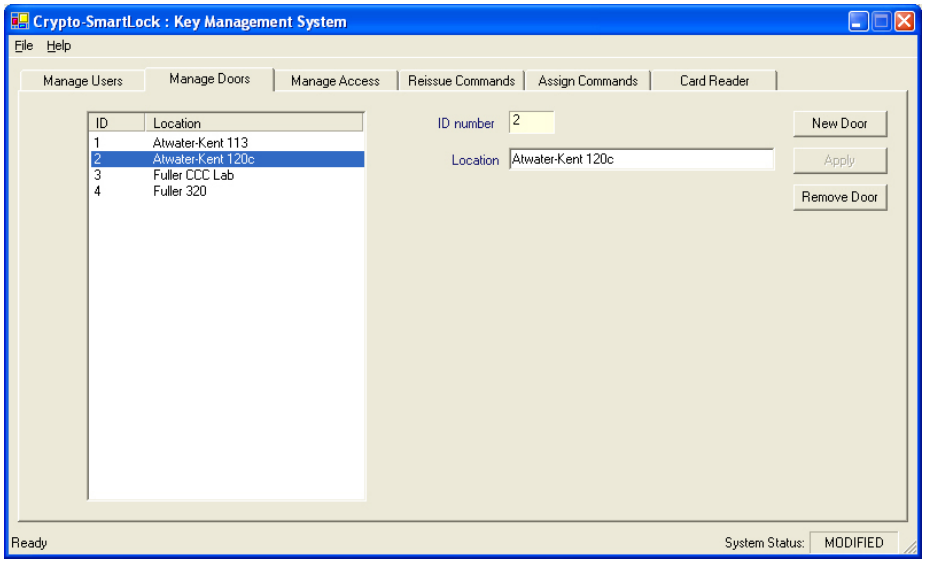

Figure 6.4: Manage Doors Tab Screenshot of the GUI

The *Manage Doors* tab, shown in Figure 6.4, is very similar in design to the *Manage Users* tab. For each new door lock module that is installed on a door, an entry needs to be added in this tab. Like users, doors are internally managed using a unique Door ID. To make it easier to manage things, each door may be assigned an alpha-numeric string referring to the door's location. System administrators may use any naming scheme they like, such as *AK113* to refer to room 113 in the Atwater-Kent building at Worcester Polytechnic Institute.

If a door lock module is ever physically removed from a building, it may be removed from the Key Management System by selecting the door from the list,

and clicking the *Remove Door* button. No commands need be issued in this case, as it is assumed that the door ceases to exist.

Manage Access

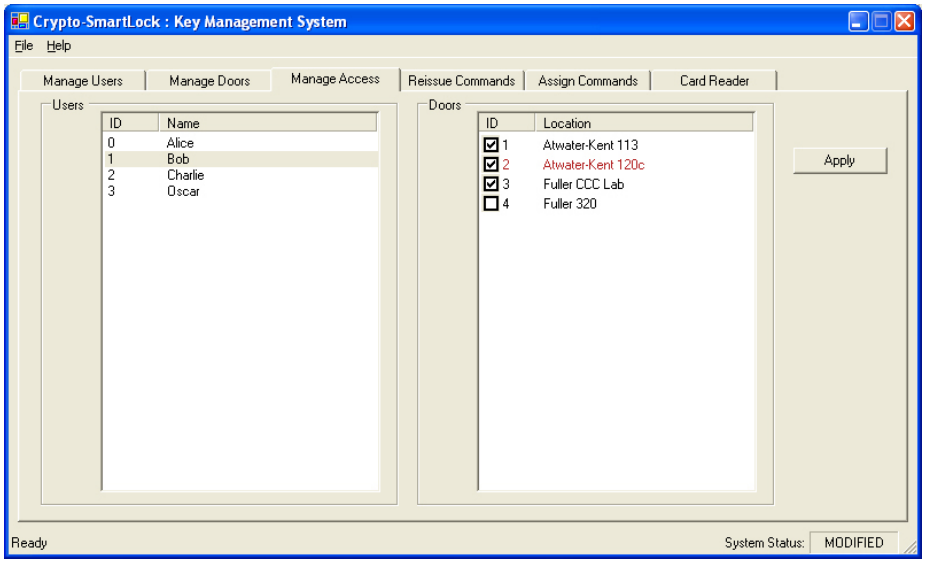

Figure 6.5: Manage Access Tab Screenshot of the GUI

Once some users and doors have been added to the Key Management System, users may be given access to doors. This is done in the *Manage Access* tab. Access Permissions are managed according to users, such that each user may be granted or denied access to any door in the system. An administrator may select a user from the list at the left, and mark check boxes next to every door, in the list at the right, that they would like to grant the user access to. Every time access to a door is denied or granted, the GUI will show the door in red to denote that a change has been made. Changes need to be asserted by clicking the *Apply* button. An example of this is shown in Figure 6.5.

As users are given new access permissions to doors, the system internally manages changes. For a user to have access to a particular door, a new 128-bit AES cryptographic key, known as K_{Auth}, is scheduled to be added to the user's card. This same K_{Auth} is encapsulated in a command and scheduled to be sent to the door via the user's card. By default, if a user is granted access to a door, an administrator need only select the user within the *Manage Access* tab, mark a checkbox next to the appropriate door, and synchronize the user's card with the Key Management System. If desired, the commands scheduled to be sent to doors via the user's card who is given this new access may be re-assigned to any other user's card instead, or in addition. Once the door processes this

command, regardless of which card carries it over, the user will have access to the door.

Reissue Commands

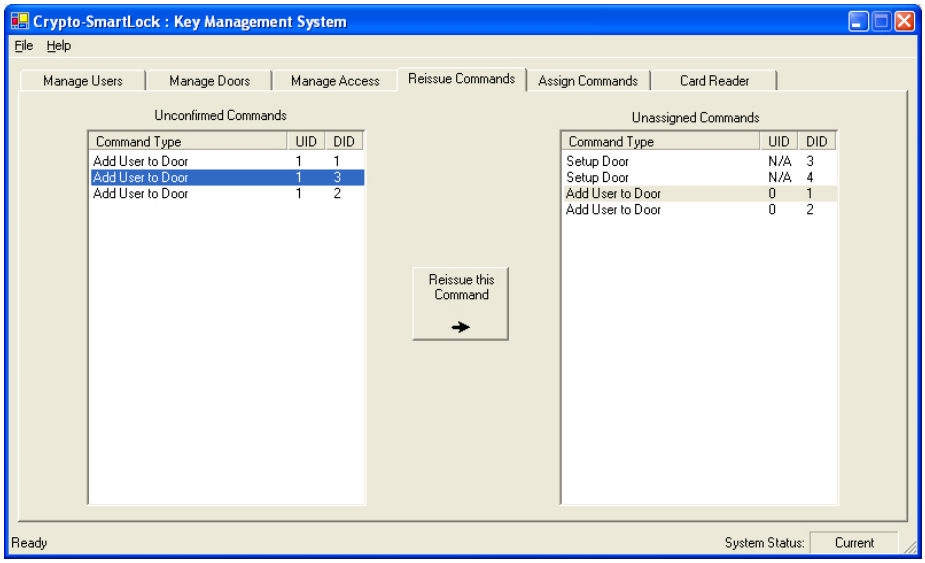

Figure 6.6: Reissue Commands Tab Screenshot of the GUI

In rare cases, an administrator may find themselves needing to reissue a command. Usually, a command is assigned to a user's card, brought to a door, and processed by the door. When this happens, the door issues a confirmation onto the card, which the Key Management System can receive and keep track of. Until a confirmation is brought back to the Key Management System from a door, each command is internally marked as being unconfirmed. Any command that was already copied to a user's card, but has not yet been confirmed, may be reissued using the *Reissue Command* tab, as seen in Figure 6.6. Once an unconfirmed command is reissued, it may be assigned to any user. Administrators should only need to use this functionality in specific circumstances, such as when a user's card containing a command has been lost.

Assign Commands

To synchronize the state of every door lock module with the state of the Key Management System, commands must be transfered to each door. By default, if a user is being given access to a door, their card is assigned the command to give them access to that door. In most other cases, though, an administrator must manually assign commands to a user. It is up to the administrator to

choose which users will transfer important commands, such as those to setup a door for the first time, or remove access for a user who has been fired. Full control over command assignment is provided via the *Assign Commands* tab of the GUI. An example of its usage is shown in Figure 6.7.

Commands that have not been moved from the Key Management System to a user's card may reside in two categories: assigned to a user, unassigned, or both. Every command, except adding a user to a door, is unassigned by default. To assign any of these commands, an administrator may select any user from the list at the left, then select any command from the list at the right, and use the *Assign Command to this User* button. Any of these commands may also be copied to a user's card – that is, assigned to the user and left in the unassigned list, so that it may be assigned to another user as well. This may be done using the *Copy Command to this User*.

Once a command is assigned to a user, it is scheduled to be copied to their card the next time it is synchronized with the Key Management System. Any command assigned to a user may be unassigned by selecting it and clicking the *Remove Command from this User* button.

Card Reader

The last tab in the GUI is called the *Card Reader* tab. This is where all communication between the Key Management System and physical key cards occur. Once a card is inserted, an administrator may click the *Initialize Card Reader* button to initiate communication. The Key Management System will execute a Handshake function on the card to determine what ID it is, and how many confirmations it is carrying. The ID returned corresponds to users in the key management system, so once the reader has been initialized, the *Card Reader* tab will display information about the user that the card belongs to. An example of this may be seen in Figure 6.8.

If the card is new, the tab will display that it is a Blank Card. Blank Cards may be assigned to any user in the system who does not have a card assigned to them yet. A list in the lower left corner of the tab shows all users who have not been assigned a card. If any incompatible cards are inserted and the *Initialize Card Reader* button pressed, the Key Management System will display an error message.

For cards already assigned to a user, the *Card Reader* tab will display all commands scheduled to be copied over to the card. Once the *Synchronize with Card* button is clicked, all scheduled commands will be copied over to the card. At the same time, any confirmations on the card will be returned to the Key Management System, and displayed in a list at the lower right hand corner of the tab. Once these command confirmations are received, the corresponding commands are internally marked as confirmed.

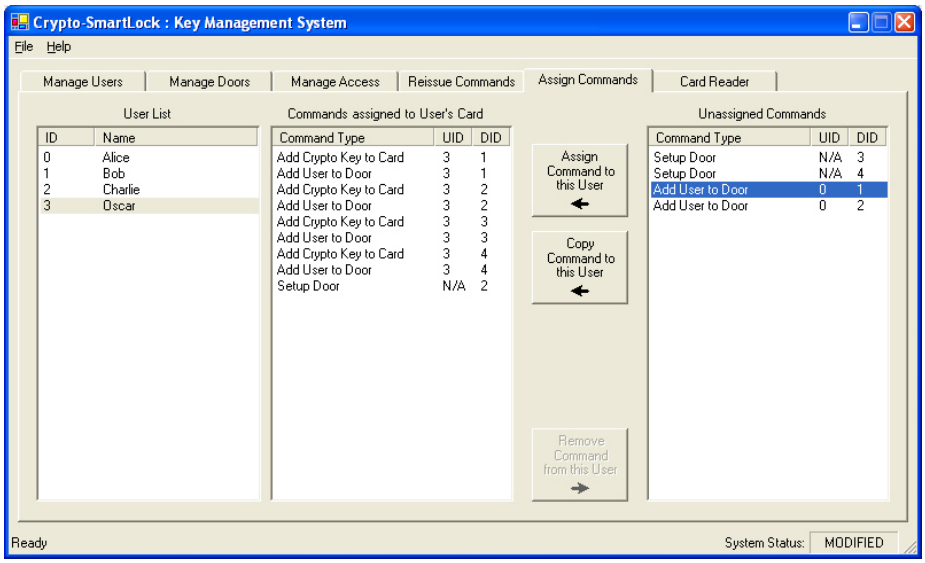

Figure 6.7: Assign Commands Tab Screenshot of the GUI

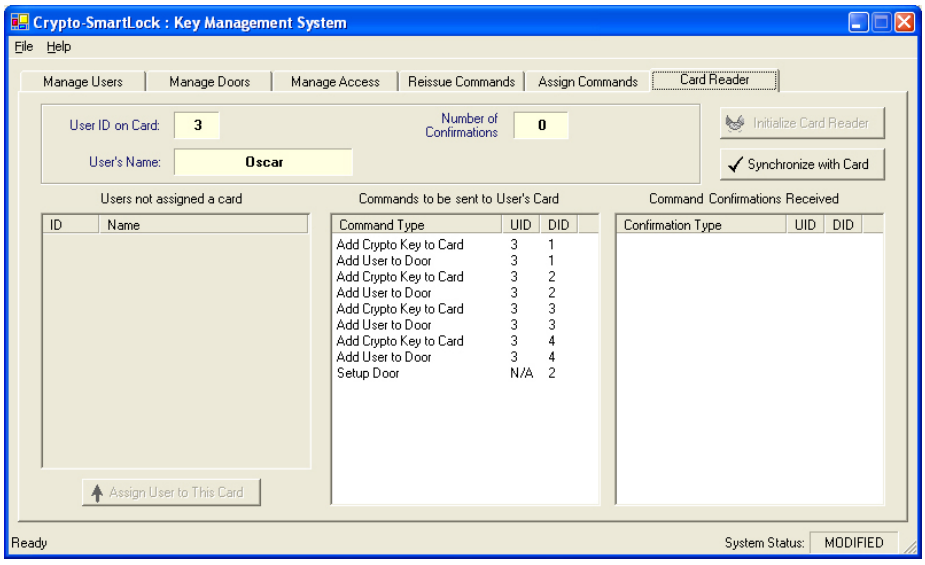
Crypto-SmartLock : Key Management System
File Help
Manage Users
Manage Doors
Manage Access
Reissue Commands
Assign Commands
Card Reader
User ID on Card: 3
User's Name: Oscar
Number of Confirmations 0
Initialize Card Reader
Synchronize with Card
Users not assigned a card
ID Name
Assign User to This Card
Commands to be sent to User's Card
Command Type UID DID
Add Crypto Key to Card 3 1
Add User to Door 3 1
Add Crypto Key to Card 3 2
Add User to Door 3 2
Add Crypto Key to Card 3 3
Add User to Door 3 3
Add Crypto Key to Card 3 4
Add User to Door 3 4
Setup Door N/A 2
Command Confirmations Received
Confirmation Type UID DID
Ready
System Status: MODIFIED

Figure 6.8: Card Reader Tab Screenshot of the GUI

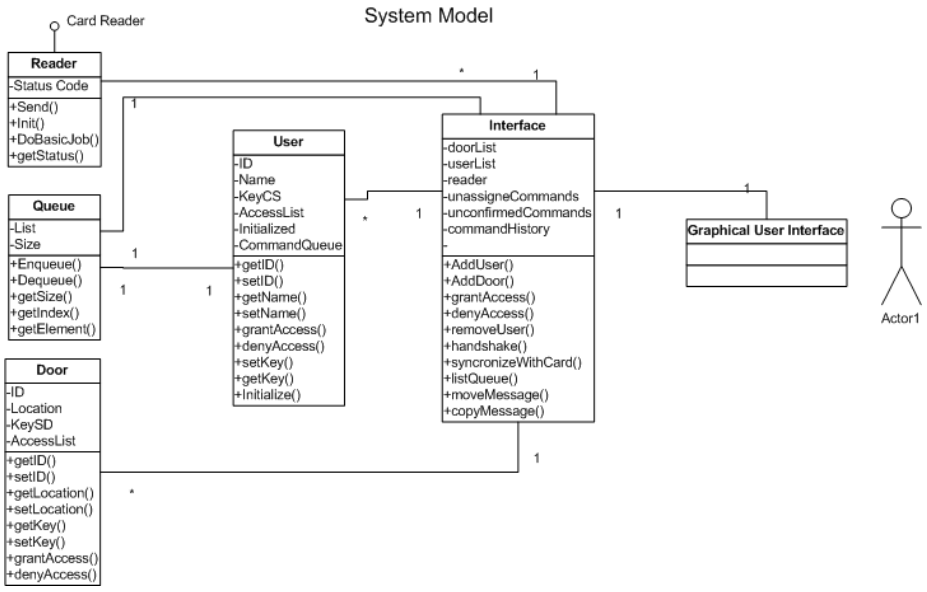

Figure 6.9: Key Management System Class Hierarchy

6.3.2 Class Hierarchy

This section will give a detailed overview of the Key Management System as a whole, specifically the class structure which are used as building blocks to the system. For an overview of the class hierarchy refer to Figure 6.9 on page 53.

At the lowest level of the class hierarchy is the class *Reader*, the class *Queue*, and the class *Door*. *Reader* is an interface to the drivers for the PC/SC Smart Card Reader. It provides all methods and status flags for communication with the BasicCard. Any time that data is sent to the BasicCard using an instance of this object, a value which was returned from the BasicCard is also returned through this object. The *Queue* class began as a First In, First Out data structure, and evolved into a First In, Any Out data structure. It stores a linked list of data structures containing formatted messages which, after encryption, are able to be directly sent to the BasicCard. The class *Door* stores all information about a single door. It provides methods for setting, changing, and retrieving all data pertaining to a particular door.

At the next level of the hierarchy is the class *User*. *User* is similar in structure to the class *Door* except that it contains all information for a particular user, and that it contains an instance of the *Queue* to contain commands destined for that particular user.

The next level houses the class *Interface* which ties all of the sub classes together. *Interface* contains lists of *User* objects, and *Door* objects, a *Reader* object, and several *Queue* objects. It also is responsible for interfacing the

Graphical User Interface with the back-end database and the BasicCard interface.

The highest level in the hierarchy is the Graphical User Interface which uses an instance of the class *Interface* to manipulate all elements in lower levels of the hierarchy.

6.3.3 Data Model

This section covers the data model for the system. It explains various methods of data storage and manipulation used throughout most of the Key Management System as well as persistent storage techniques.

User Database Model

User information is stored in memory as an indexed array of *User* Objects in the class *Interface*. Initially this array is generated statically to the maximum amount of users where all user ID numbers are initially set to a sentinel value. When a new user is created, the data for that user is stored in the array at the offset by the user ID number. When a user is removed from the system the ID of the user at the offset of the ID in the array is reset back to the sentinel value. Therefore, to determine if a user is currently valid in the system, a simple comparison of the user's ID number and the offset in the array is performed. If the ID number matches the offset in the array, the user is considered valid. When a new user is created a flag in each user object determines if the user has been assigned a card. If the flag is cleared to zero, the user will be displayed in a list in the GUI enabling the system administrator to assign that user a card.

Door Database Model

Information regarding doors is stored in memory as an indexed array of *Door* objects in the class *Interface*. This array is treated in much the same way as the user database except, since the Key Management System refers to itself as a door with the ID number of zero, the instance of the door in this array at location zero can never exist.

User-Door Access Permissions

Since the GUI must be able to determine the particular doors for which each user has access to and the conversely the particular users that have access to each door, it was necessary to maintain accurate records for each scenario. When a user is given access to a door, a the ID of the door is added to a list stored in the instance of the user, and the ID of the user is stored in a list the instance of the door.

Command Queueing

The system contains several data structures designed specifically for commands and messages destined for the BasicCard and the Door. Initially these began as a First In First Out (FIFO) buffer, but as the system evolved, the queue data structure became more of a list or collection where items were always added to the head of the list, but could be removed in any order.

One of these data structures is used to hold the system-wide unassigned commands and the system-wide unconfirmed commands. The reason for which items could be removed from the list in any order came about as a result of assigning specific commands to any card which the administrator could designate. This property was again used to remove confirmed commands from the unconfirmed commands lists as the confirmations came back from the door.

Messages destined for a particular user's card are stored in a typical FIFO buffer within the particular instance of the user in the system. When a command is assigned to a user, that particular command is enqueued in the user's queue to be processed when that particular user's card is inserted in the reader.

Persistant Storage Techniques

One of the requirements of the Key Management System is the ability to maintain program state. In order to accomplish this, the software traverses all objects in the database and writes every data field in each object to a file.

When the program is loaded, the data file is checked for consistancy and the program state is restored. If the data file has been corrupted or fails to exist in the program directory, the software will create a new instance of the database.

The program will save state at several different times. When an administrator syncronizes the Key Management System with a card, the state of the system has been modified. Since new commands or confirmations have been sent to and/or removed from the cards, the software must retain state to prevent a malicious user from concealing their efforts to issue unwaranted commands. Another time when the state of the system is saved is when the program is closed. At that time, if the database has been modified, the administrator will be prompted by a pop-up box to save the state. The administrator will then have the choice to save or discard the changes since the last save.

Importing and exporting the database is useful for backing up past revisions of the system. This will save or load the current state of the system to or from a file specified by the administrator. Since syncronization with cards will make permanent changes to the state of the system, it is unadvisable to import any data file except the latest revision. Importing an old revision will cause records of commands and confirmation to be lost, and future commands will be blocked because transaction IDs will be used multiple times.

6.3.4 Command Processing

This section describes the process of command generation. The particular aspects of the command generation system that will be described are, how and

where the commands are generated, cryptographic key generation, nonce generation, and passing parameters to the BasicCard.

When and Where are Commands Generated?

The command ADDUSER and the command ADDKEYTOCARD is generated whenever a user is given access rights to a particular door. ADDUSER is generated and enqueued to the user's command queue first and it is meant as a means of transmitting a K_{Auth} to a door. ADDKEYTOCARD is generated and enqueued second and is used to pass the same K_{Auth} to that user's card.

The command REMOVEUSER is used to deny a user access rights to a particular door. Whenever a user is denied access to a single door, this command is generated and will be used to delete the K_{Auth} from that door. This command is sent by default to the unassigned user's queue so another card, such as a janitor's can be used to remove a user from a door. This command is also used when a particular user has been removed from the Key Management System. When this happens, the user's access list is retrieved from the database, and for every door which the user has had access to, this command is generated and added to the unassigned commands list.

The next command is the SETUPDOOR command. This is generated when a new door is added to the Key Management System. It's purpose is to pass the K_{SD} to the door and set up the door ID.

Next is the NEWUSER command which is generated when a user is selected from the uninitialized users list on the *Card Reader* tab of the GUI. This command transmits the K_{CS} to the card and initializes the ID of the user on that card.

The final command is NOP which is generated to follow every REMOVEUSER command. This is a placeholder with the same transaction ID as the command issued to add the user to the door. The reason for this is explained more in-depth in Section 6.1.

How are Cryptographic Keys Generated?

There are three different types of cryptographic keys in the system that are all generated in different locations. K_{Auth} is used for communication between the card and the door. Since the Key Management System never needs to know this key, it is never retained. When the ADDUSER and ADDKEYTOCARD commands are generated, K_{Auth} is also generated using random unsigned characters. This key is included in both of the commands and is never recorded by the Key Management System. K_{CS} and K_{SD} are generated differently from K_{Auth} but in a similar manner to each other. They are both generated as an array of unsigned characters within the constructor function of the *User* and *Door* classes respectively.

What is a Nonce and How is it Generated?

A nonce is a random piece of data which is used to prove the validity of a message.[4] In the protocol we designate that nonces used within this system are 16-Bits long. In every command which is to be decrypted by either the BasicCard, the Key Management System, or the Door, a random nonce is generated and encrypted by the sender, and decrypted and returned to the sender by the receiver. The sender then will compare the sent nonce with the received nonce and if they match, then the sender will know that the receiver was able to successfully decrypt the message. Every encrypted transmission being generated on the Key Management System contains a random nonce as bytes three and four.

How are Commands Sent to the BasicCard?

When the administrator attempts to synchronize the Key Management System with the BasicCard, the system begins processing the user's queue and each element in the queue is sent in turn to the BasicCard. The first step in this process is fetching each command from the user's queue. Each command is then parsed and the appropriate cryptographic keys are fetched from the database. Each block of the message is then encrypted and passed to an instance of the *Reader* class. In the *Reader* class, the original opcode which accompanies the data packet is used to determine the appropriate flow of execution. Functions defined on the BasicCard are determined by two unique bytes, the CLA byte, and the INS byte. When the Key Management System is ready to send a message to the BasicCard, the opCode determines what values of CLA and INS are used, and what size data packet is being sent to the card. Because every function on the BasicCard uses a pass-by-reference scheme, for every message sent to the BasicCard, a return value of equal size to the message sent is returned to the Key Management System. This return value is used whenever the Key Management System needs to retrieve information from the BasicCard, for example, when it is receiving a confirmation.

6.4 BasicCard Hardware

Our final implementation of BasicCard code takes up 3408 bytes out of the total 8096 bytes available on the ZC3.9 Enhanced BasicCard. In addition to this, there are two permanent values that are written to EEPROM when a card is initialized: the two-byte card ID and the 16-byte cryptographic key: k_{cs}. This leaves 4670 bytes to store data.

For each door a user requires access to, the BasicCard must store a 16-byte AES key. When commands need to be transfered to a door via a BasicCard, that will require much more storage space. Each command is composed of 34 bytes, which includes a 32-byte encrypted command, and a 2-byte unencrypted nonce.

The filesystem implemented on the Enhanced BasicCard adds some overhead to the base storage requirements listed above for each command and cryptographic key to be stored in EEPROM. There are three categories of files that are stored on the BasicCard: confirmations, commands, and cryptographic authorization keys. For each single file stored on the BasicCard, 19 bytes of overhead are added. The confirmations file will only exist when some commands are processed and removed from the card, so it does not need to be taken into account when calculating how many commands or authorization keys may be stored.

For each single door a user has access to, 19 bytes for file overhead, plus 16 bytes for the cryptographic key are used. The 19 bytes of overhead include the door ID, so no extra space is needed for it. This totals 35 bytes for each door a user is given access to. If no commands reside on a card, a total of 133 authorization keys may be added.

Each command on a card is stored in a file that refers to the door it will go to. This means that 19 bytes of overhead are added for each door that any number of commands are assigned to. Each single command adds 34 bytes in addition to that. As an example, if there are no authorization keys on a card, and all commands are assigned to a single door, 136 commands will fit. In a more realistic example, an equal amount of commands are assigned to 24 different doors. While the first example only added 19 bytes of overhead, 24 doors will add 456 bytes of file overhead. That leaves space enough for five commands per each of the 24 doors.

In most cases, normal user's cards would only store authorization keys to allow them access to doors in the system. A user issuing commands would probably also have access to many doors, so space on the ZC3.9 BasicCard is very limited. If this project was implemented the same way on the Professional ZC5.5 BasicCard, EEPROM space for authorization keys and commands would be increased greatly. In addition to simply having a larger EEPROM, about 2 kilobytes of program space would be freed, since AES support is built into the operating system on the Professional BasicCard ROM.

6.5 BasicCard Software

The design of the BasicCard portion of the software was relatively straightforward. The design of the card software is almost entirely dictated by the protocol design and the functionality of the BasicCard operating system.

BasicCard programs consist of a set of commands. Each command is identified by two bytes called CLA and INS. BasicCard commands are called from both the key management system and the door system. Most commands are called from one or the other, but not both. Each command is structured as a Basic subroutine. The arguments are passed in to the subroutine, and then the results are passed out.

The BasicCard operating system provides a file system, which is used to store the commands and keys.

6.5.1 BasicCard Commands used by both Door and Key Management System

handshake (doorID, cardID , numCommands)

This function is the only one that's used by both the central system and the door. The doorID is passed in, with the central system sending the door ID of 0x00. The card looks up the number of commands or confirmations for this door, and the cardID and number of commands are returned. The card remembers which door it is talking to for the remainder of the session.

6.5.2 BasicCard Commands used by the Door

authenticate (challenge)

The authenticate function is the heart of the BasicCard part of the system. This is method for actually gaining access to a door. The door generates a 16 byte random challenge,encrypts it with K_{Auth}, and passes it in to this command. The BasicCard decrypts the challenge, increments it by one, and re-encrypts it. This value is returned to the door to prove that the card has the proper key.

getCommand(encryptedBlock)

This function returns the 32 Byte encrypted block which contains the command. It returns the first command in the file for the current door.

confirmCommand(encBlock ,nonce ,confNonce)

This function is used to delete a command and store the confirmation for it. The encBlock is the confirmation itself. Nonce is the nonce from the command being confirmed. confNonce is a nonce for the confirmation. The card compares nonce to the nonce from the current command. If the nonces match, the card deletes the command and stores the confirmation. The contents of the confirmation are detailed in Table 6.7.

Table 6.7: Contents of Confirmation

Index	Value
0	0x06
1	nonce (h)
2	nonce (l)
3	transactionID (h)
4	transactionID (l)
5	DoorID

6.5.3 BasicCard Commands used by the Key Management System

getConfirmation(encBlock, doorID)

This command returns the confirmation as detailed in table 6.7. The doorID is also returned, so the key management system can look up the appropriate decryption key.

deleteConfirmation(nonce)

This command takes the nonce from within the encrypted confirmation. This proves to the card the the key management system successfully decrypted the confirmation, which allows the card to delete it.

putCommand(encBlock1, encBlock2, nonce)

This command is used to give the card a command to carry to a door. encBlock1 is encrypted with the key between the system and the card. Its contents are detailed in table 6.8.

Table 6.8: Put Command Contents

Index	Value
0	nonce to verify card identity
1	nonce to verify card identity
2	plaintext of nonce for command
3	plaintext of nonce for command
4	target doorID
5	cardID
6	cardID

encBlock2 is encrypted with the key between the system and the door and contains the command itself, as detailed in table 6.1.

The nonce to verify card identity is then returned in the nonce parameter so the key management system can verify that it gave the command to the correct card.

setKey(encBlock1, encBlock2, nonce)

This command is used to set a K_{Auth} on the card. The contents of encBlock1 are detailed in table 6.9.

The card then returns the plaintext of the nonce so the key management system can verify that the key was given to the correct card and properly decrypted.

Table 6.9: Set Key Contents

Index	Value
0	0x07
1	nonce(h)
2	nonce(l)
5	doorID
6	UID(h)
7	UID(l)

setupCard(encBlock1, encblock2, nonce)

This command is used to initialize the card. The card's new ID and K_{CS} are stored in the EEPROM.

The contents of encBlock1 are detailed in table 6.10.

Table 6.10: Setup Card Contents

Index	Value
0	0x09
1	nonce(h)
2	nonce(l)
6	UID(h)
7	UID(l)

EncBlock2 contains the new K_{CS}. The card then returns the plaintext of the nonce.

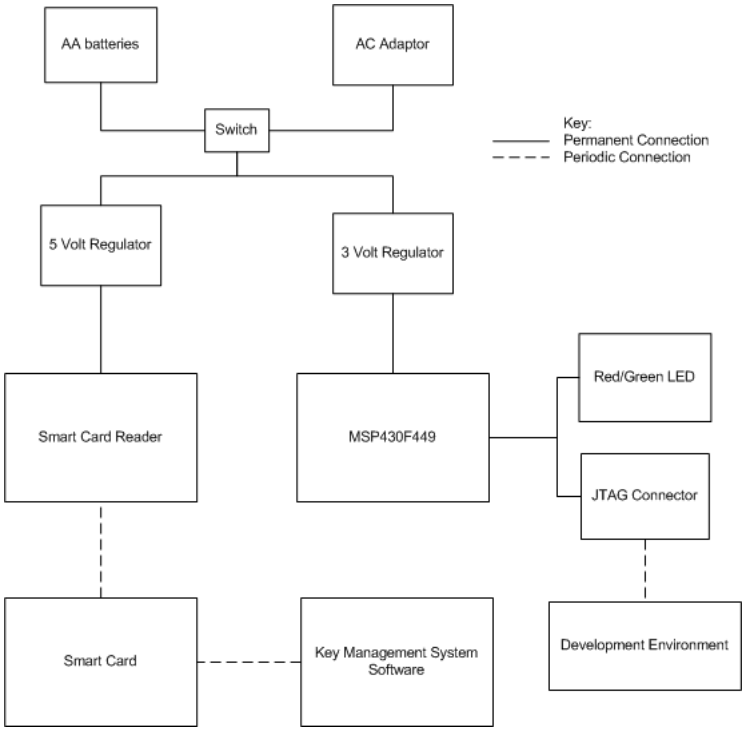

Figure 6.10: Prototype Block Diagram

Figure 6.11: Crypto-SmartLock Prototype - Outside View with JTAG interface

Figure 6.12: Crypto-SmartLock Prototype - Outside View with Power Connector and Switch

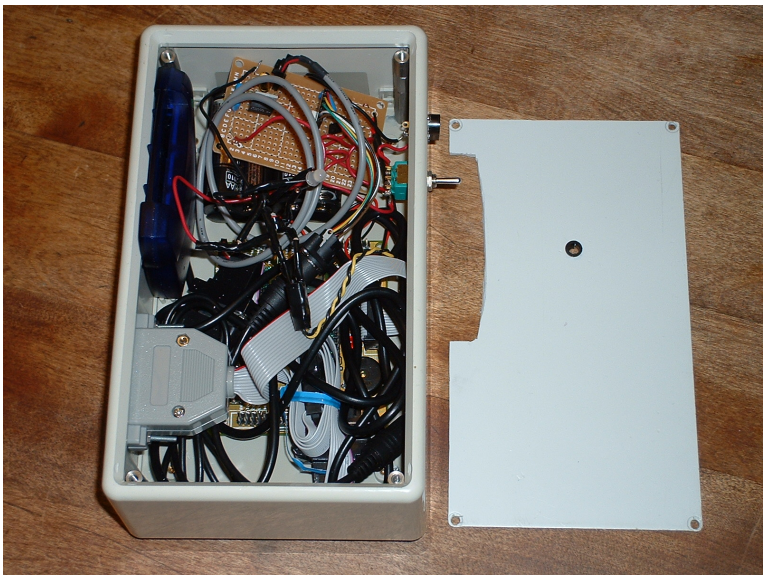

Figure 6.13: Crypto-SmartLock Prototype - Inside

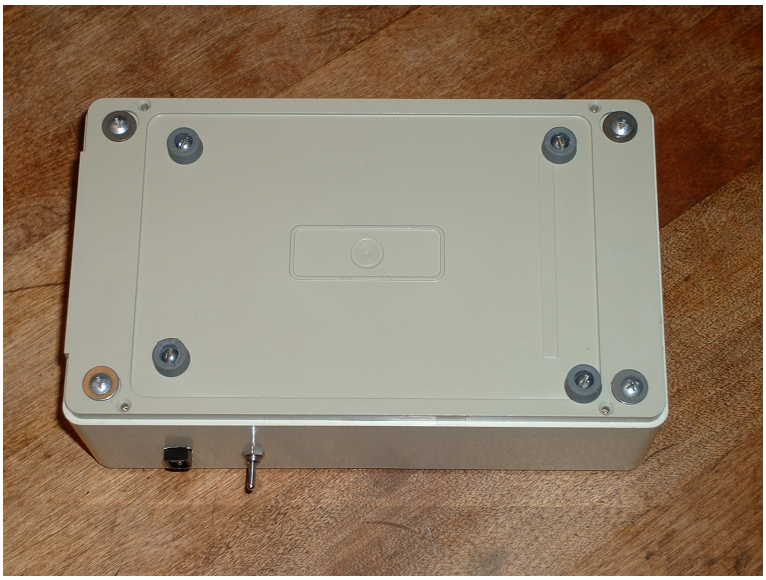

Figure 6.14: Crypto-SmartLock Prototype - Bottom with Rubber Feet

6.6 Door Hardware

The combination of the SparkFun MSP430 development board and the ACR-30S serial Smart Card reader formed the core of our prototype hardware. The self-contained prototype required several auxiliary components; the interaction between them is shown in Figure 6.10. The most important was power and power circuitry. The MSP430 and card reader require different voltages, 3.3V and 5V, respectively. Thus, voltage regulators were needed. An MC2805 regulator worked well for the MSP430; an LM317 variable regulator set to 3.37V supplied the proper voltage to the card reader. The voltage input to these regulators can be selected from four AA-size batteries (6V total) for normal operation, or a 7V wall-outlet plug for development. The primary output device of the lock is an LED that can display red, green, or orange; this is connected to one of the MSP430's digital I/O ports. A beige plastic enclosure houses these components (Figure 6.13), along with the JTAG parallel-port connector for MSP430 code development (Figure 6.11). The enclosure also has a toggle switch for selecting between the internal batteries or external power connector (Figure 6.12), and rubber feet to prevent scratching of surfaces (Figure 6.14). The internal power and communication connectors are modified personal computer cables. The MSP430 development board and card reader communicate via an RS-232 serial port; the cable between them is composed of two spliced RS-232 ports from an old computer case. Likewise, the card reader's power connector is a PS/2 keyboard cable, so it connects to its voltage regulator via an old motherboard's PS/2 port. Lastly, the MSP430 board has a power connector identical to that of a motherboard processor fan, so a cable from an old fan is used. A schematic of all of these components is shown in Figure 6.15.

6.6.1 Human Interface Design

The Crypto-SmartLock Human Interface Design for operating a door lock is intentionally simple. Not only is this a result of a desire for intuitive use, it is a result of cost considerations and other practical concerns in interfacing to the Texas Instruments MSP430. In addition to this, the simplest interface on door locks lends to better security, as malicious users are only given limited feedback and interaction with a door lock.

Table 6.11: Door Lock Human Interface Design

Event	**LED Indication**
Processing Commands	Light Orange until processing complete
Command Error	Flash Red and Orange alternatively for 1/4 second each, and two seconds total
Access Denied	Blink Red on and off for 1/2 second each, three times
Access Granted	Light Green for one second

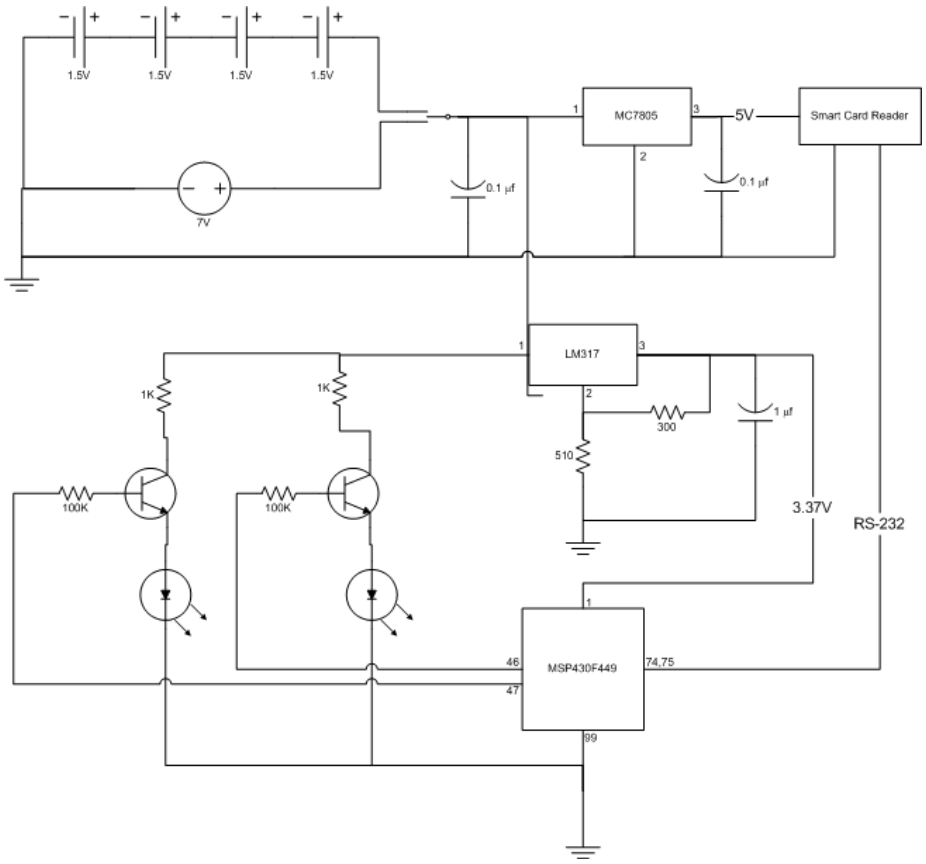

Figure 6.15: Prototype Schematic

A door lock uses a single tri-color LED to provide visual feedback to the user. This LED is always in one of four states: unlit (off), green, red, or orange (a combination of green and red). Whenever a user inserts a key card into a door lock, the door will first check for commands on the card. If there are commands, it will attempt to retrieve and process them. After that, it will attempt to grant the user access to the door. Table 6.11, on page 65 shows what LED light patterns correspond to each of these events.

6.7 Door Software

The door software is primarily concerned with communicating on the serial port to the Smart Card reader, performing encryption/decryption operations, and storing/retrieving data from the flash memory. Serial port communication is straightforward due to the MSP430's easily configurable UART, and strings of transmitted/received data can be stored in byte arrays via the C language. Cryptography and random numbers are provided by public libraries, and access to the flash memory can be handled via subroutines whose necessary functionality is described by the MSP430 datasheet.

The main flow of the door software is an event processing loop. The door waits until it receives a message on the serial port from the card reader. If the message is a card insert status message, the door begins its interaction with the card. Otherwise, the loop continues.

The first command sent to the card reader is the reset card command. This applies power to the BasicCard and allows it to start executing commands. The door then executes the handshake command, which informs the card of the door id and gets the card's id and the number of commands pending for this door.

For each of the pending commands, if any, the door retrieves and processes the command. If the command succeeds, the door places the confirmation on the card.

After command processing is completed, the door attempts to authorize the card for access by running the authenticate card command. If this succeeds, the door will display the access granted light pattern and, in the production version, activate the solenoid.

Finally, the door sends the card power off command to the reader. This shuts down the card, and deactivates its power.

6.7.1 Cryptography on the Door

The communications between the card and door are encrypted with the AES protocol, as described in section 6.1.6. The door system uses an implementation of AES based on the original optimized ANSI C implementation by Vincent Rijmen. This version has been modified to work efficiently on the MSP430.

AES is used in the ECB mode, because it is mostly used to encrypt single random blocks for the authentication functionality. The ECB mode simplifies

the implementation, and provides sufficient security in the absence of bulk encryption tasks.

6.7.2 Serial Communication

The serial communication on the MSP430 is implemented as a hierarchical series of functions. These functions build up from those which read and write single bytes from and to the port, to functions which exchange entire messages with the card reader.

BasicCard communication was implemented by writing a communication toolkit from the ground up. The toolkit began with the serial communication functionality, and extended to contain all functionality required to communicate with the BasicCards. The kit implements various protocol components necessary for communication with the card reader. The toolkit translates from the messages generated by the 430 to the wire protocol used by the card reader. This wire protocol consists of sending ASCII text containing the hexadecimal values of the bytes to be sent. While this protocol is rather roundabout, it does simplify debugging by allowing the serial communication to be human readable without the necessity of a debugger. The toolkit also implements the checksumming used to verify communication with the card reader. The checksum function used by the Smart Card interface is a simple XOR.

In the reverse direction, the toolkit translates the message back from the wire protocol and verifies the checksum. Following this, it parses the returned status information. The returned status and data are then passed back to the caller in a C structure.

6.7.3 Flash Memory

It is necessary for the door system to store all of its state in non-volatile memory, so that it is retained during temporary power loss, such as battery replacement. This is accomplished by taking advantage of the MSP430's in circuit programmable flash memory. The same flash that is used for the program storage can also be used for non-volatile data storage.

The list of authorization keys is stored in the flash memory. Because the built in flash is part of the 430's normal address space, reading from it is no different than reading from the device RAM. The key list is stored as a C array, which through the use of special compiler directives, is allocated in the flash. This allows the list to be accessed for reading using standard C array indexing. This greatly simplifies the code compared to what would be necessary to deal with an external non-volatile memory.

Writing to the flash is slightly more complicated than reading. When the MSP430 flash is erased, it is filled with binary ones. Normal writes to flash can cause transitions from binary ones to binary zeros, but they can not reverse the process. To write to a currently erased section of flash, the program first activates the flash controller for writing, then performs the write, finally deactivating the flash controller. The actual write can be performed using any

C method of writing to memory, such as variable assignment or the memcpy family of functions. Writing to a section of flash with existing data is somewhat more complicated, as flash can only be erased in 512 byte blocks. To rewrite a value in a currently used flash block, the program copies the block to the device RAM, erases the block, modifies the data as necessary, and writes the modified data back to the flash block. The erase is performed by activating the flash controller for block erase and performing a dummy write anywhere in the block.

The MSP430 Flash must store the K_{Auth} for each of the 1024 possible users. With each K_{Auth} taking up 16 bytes, this requires 16 kilobytes of storage. The flash must also store the vector containing which transaction IDs have been used. There are 2^{16} possible transaction ID's so, 2^{16} bits, or 8 kilobytes are required. This is a total of 24 kilobytes of non-volatile storage needed. The program code for the door also requires approximately 30 kilobytes of storage. This leads to a total Flash requirement of 54 kilobytes, which is only available in the 60 kilobyte flash MSP430 devices.

6.8 System Flow

6.8.1 Adding a user or door to the system

When a user or door are added to the Key Management System, several events actually take place. First entries for both doors and users are added to the internal database and ID numbers are automatically generated by the software. If a door has been added, commands are generated to set up the door with a new cryptographic key, K_{SD} and the door ID number. These commands are placed in the unassigned commands data structure. If a door was added to the database, an administrator will next need to assign the commands to a specific user and synchronize the system with that user's card. If a user was added, a blank card will need to be initialized for that user. The process for adding a user is described in more detail in Figure 6.16, and the process for adding a door is described in Figure 6.17.

6.8.2 Command Passing

Figure 6.18 describes the overall system flow from a card being initialized through command execution on the door. This flowchart shows the implementation of the protocol and how it is used throughout our project. The details of command passing on the BasicCard are described in Figure 6.19, and the details of the command passing on the Door are described in Figure 6.20.

When a command is passed to the BasicCard it is stored in a file based on the target door ID number. When a new door is created the command to setup the door is stored in a file based on the sentinel value. This will not allow more than one SETUP DOOR command per card.

When the card is inserted in the door, the door requests all commands from the card destined for that door. The door will fetch and execute all commands

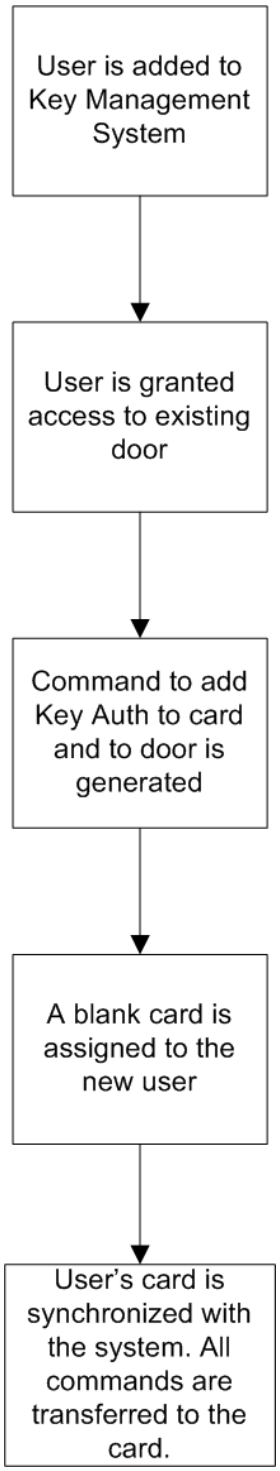
User is added to Key Management System
User is granted access to existing door
Command to add Key Auth to card and to door is generated
A blank card is assigned to the new user
User's card is synchronized with the system. All commands are transferred to the card.

Figure 6.16: Flowchart for adding a user to the system

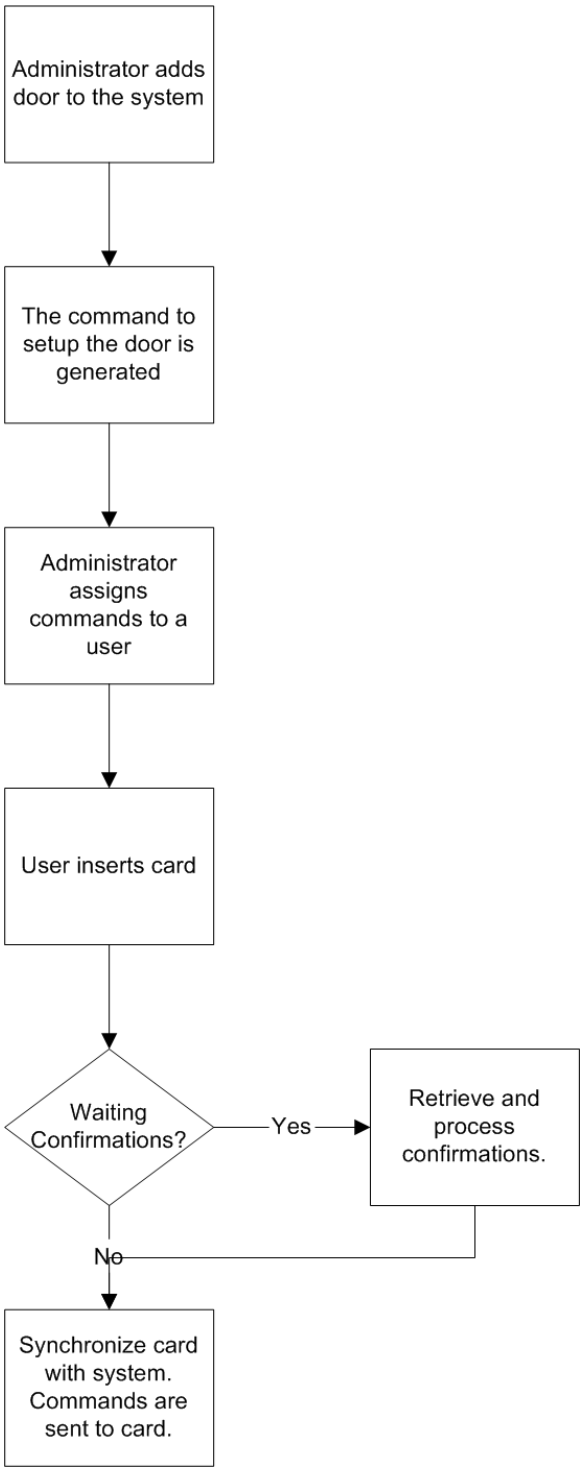
Administrator adds door to the system
The command to setup the door is generated
Administrator assigns commands to a user
User inserts card
Waiting Confirmations?
Yes
Retrieve and process confirmations.
No
Synchronize card with system. Commands are sent to card.

Figure 6.17: Flowchart for adding a door to the system

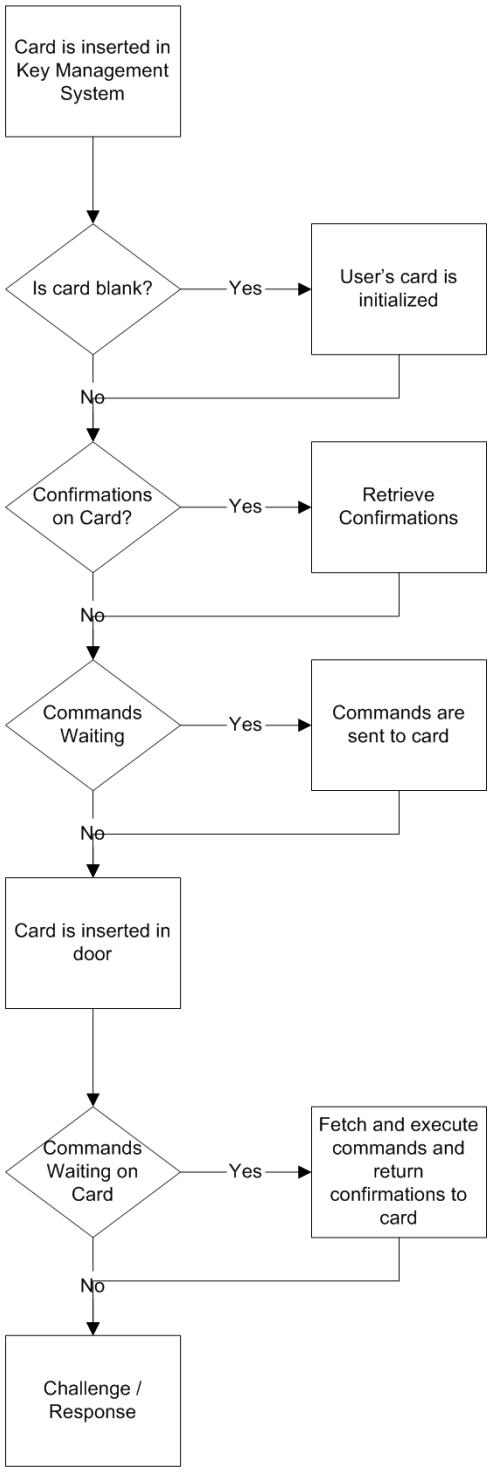

Figure 6.18: Flowchart for command passing

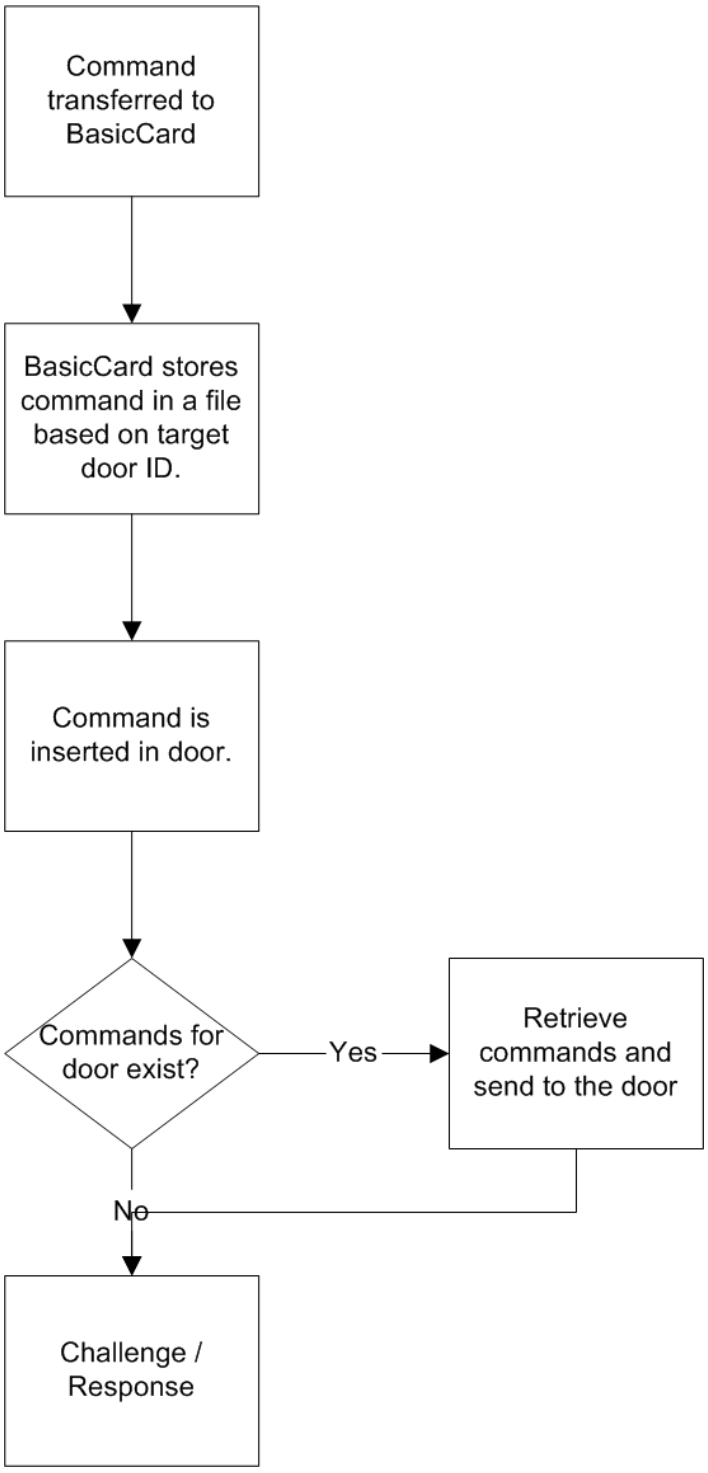
Command transferred to BasicCard
BasicCard stores command in a file based on target door ID.
Command is inserted in door.
Commands for door exist?
Yes
Retrieve commands and send to the door
No
Challenge / Response

Figure 6.19: Flowchart for commands on BasicCard

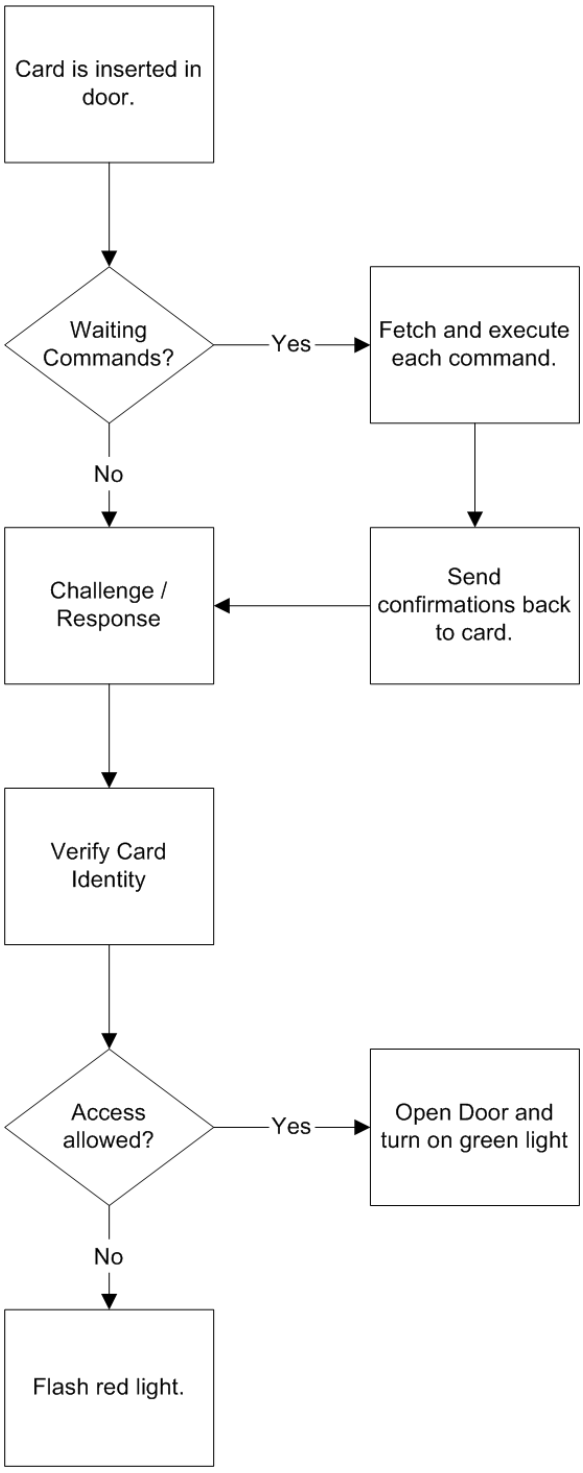

Figure 6.20: Flowchart for commands on door

stored in the particular file on the BasicCard and then attempt a handshake with the door. As each command is executed, a confirmation message is placed back on the card to let the Key Management System know that the command has been executed.

6.9 Attack Scenarios and Prevention

The overall system has been designed to prevent any reasonable attack upon its components. Every known attack has been addressed, from the most obvious to the more obscure and subtle. Listed below are possible attacks that a malicious user might attempt, and how the system is set up to prevent them. Also listed are the handful of attacks that the system does not prevent, and the reasons for that point of insecurity. As is common in information security documents, the hypothetical attacker is referred to as Oscar below.

6.9.1 Attacks Prevented

Oscar creates a false card to open a door

Oscar does not know K_{Auth}, the cryptographic key used between a card and a door used to grant access. Even if Oscar knows the card number of a valid card that has access to a door, without K_{Auth}, his card can only supply a random or fixed value as a response to the door's challenge. The number of possible responses is 2^{128}; the best approach, combinatorially speaking, would be to pick one response value and keep trying it until access is granted. However, in the average case, this would take the attacker billions of billions of times longer than the age of the universe, and is not a viable option.

Oscar attempts to extract information directly from a real card

It is possible to steal a legitimate card and extract information from its EEPROM, such as authorization keys, and the card's K_{CS}. However, this requires a clean room and hundreds of thousands of dollars worth of equipment. Not only is this impractical, but by the time Oscar learns the keys from the card, the original user will most likely have noticed his card missing and gotten a new one, with a new set of cryptographic keys. If this is Oscar's own card, he will not learn any information from the card that would benefit him in any way, since his authorization keys are only valid for himself, and the K_{CS} can only be used to put false commands on his own card (which will not be accepted by any door without the valid K_{SD}).

Oscar attempts to extract information directly from a door

Oscar could try to dismantle a door lock and learn authorization keys from the microcontroller at its core. However, if he is able to do this, he would be able to break into the door in the first place. In addition, if he removed the

microcontroller, he would need to carefully remove it from the door circuitry and take it to a location in which he could examine the flash memory. Even if he were successful, the break-in attempt would be obvious during this time, and the door could be replaced with a new set of encryption keys.

Oscar creates a false door to extract a K_{Auth} from a card

Oscar knows that a card's response to a door will be the door's challenge plaintext incremented by one, encrypted with K_{Auth}. Because he knows this relationship between the challenge and the response, a recognizable plaintext attack is possible on the ciphertext. Oscar would create a false door that identifies itself with a real identification number and sends a random challenge to the card. Oscar can then attempt to try all possible AES keys on the challenge/response pair to see if the plaintext is a value and the value, incremented. In the event that more than one key is possible, Oscar can try the keys on another challenge/response pair to see if the plaintext is valid. However, two factors make this approach. Oscar must be able to take a user's card without the user noticing; otherwise, the user could simply have his/her card, and all the keys within, invalidated. In addition, even with the most powerful computing equipment available, a brute-force attack on AES may take billions of years.[6] Barring a new discovery of a serious vulnerability in AES, this attack is not viable.

Oscar creates a false door to extract a K_{SD} from a card.

Oscar might try to examine an encrypted command to determine the key between a door and the key management system. His false door could accept a valid command from a card, which he could then analyze. Since a command will contain known quantities such as a door number and one of a limited number of opcodes, a known plaintext attack is possible. As with obtaining K_{Auth}, however, a brute-force attack on AES is not possible within a reasonable time-frame even if a card could be stolen. As a result, Oscar cannot obtain K_{SD}.

Oscar creates a false key management system to extract a K_{SD} from a card

In the event that a stolen card is known to contain a confirmation message, but not a command, Oscar might try to get the confirmation through a false key management system and determine K_{SD} from that. Once again, however, the resilience of AES against a brute-force attack prevents this.

Oscar creates a false key management system to extract a K_{CS} from a card

A card identifies it to the key management system in the same manner as it does with a door. As a result, the standard challenge/response handshake is used, except using K_{CS} instead of a K_{Auth}. Fortunately, because this protocol is the same, a false key management system does not open up any vulnerabilities that

a false door would not. The integrity of the cryptographic algorithm makes the key unattainable.

Oscar steals a card to prevent a command from reaching a door

If Oscar takes a user's card, any commands on it will not reach the door. Fortunately, the key management system retains unconfirmed commands until they are confirmed. If a card goes missing, the command can be given to another card for transmission, and the original user's card can be re-created.

Oscar steals a card to prevent a confirmation from reaching the key management system

In the event that a command is executed, but the card supposed carrying the confirmation goes missing, the key management system can re-issue the original command. When it reaches the door, the command will not execute, since its transaction number was already recorded on the door; thus, the key management system will know for sure the command was properly carried out. If the command was in fact not carried out for some reason, it will still execute at that point.

Oscar creates a false door to prevent a command from reaching a valid door

Oscar might want to prevent a command from reaching a door without immediately raising the suspicions of the system administrators. If he could take the card containing the target command and supply it with a false door, the card might consider the command to have been executed. However, each command contains a random value, or nonce; this is the only piece of information within a command that a card knows. The door must send back the nonce to the card to confirm it is in fact the proper door; thus, Oscar's attack will not achieve the desired result. Oscar could try to use a randomly-selected nonce to return to the card in the hopes that it happens to be the correct one, but his chances of success are one in 2^{16}. Depending on the speed of Oscar's false door, trying enough nonces to find the correct one for one command will take several hours to several days, on average. By this time, the card will most likely have been noticed missing. If not, however, and Oscar did find the right nonce, his door would not be able to encrypt the confirmation message properly, and the key management system would alert the system administrators that the command failed and should be re-issued.

Oscar creates a false key management system to prevent a confirmation from reaching the real key management system

Blocking a confirmation is of dubious value, as it will only alert system administrations to a problem, but it is nonetheless prevented to ensure proper operation of a system. The nonce method that prevents a false door from claiming to have

executed a command is also in place for the key management system. The card expects the nonce present in the confirmation from the key management system before it removes it; if a false key management system tries to remove a confirmation, it will not have the nonce.

Oscar attempts to replay a command to a door

Oscar might have acquired the ciphertext of a command, either through stealing a card and observing its interaction with his false central system, or somehow observing the interaction between a real card and a real door directly.

Oscar attempts to execute an ADD USER command after the corresponding REMOVE USER command is executed

In some cases, the confirmation for an ADD USER command may not make it back to the key management system before that user needs to be removed from the door. As a result, the REMOVE USER command might be issued and executed before the corresponding ADD USER command arrives at the door. Thus, the user will be granted access after his privileges should have been removed. To protect against this, every time the central system issues a REMOVE USER command, it also issues a NOOP command with the same transaction number as the original corresponding ADD USER command. When they arrive at the door, the transaction number of the NOOP command will be recorder, and thus the extant ADD USER command can never be executed. If it already has been, then the NOOP command has no effect, and everything proceeds as usual.

Oscar creates a false card to supply false commands to a door

If Oscar is especially malicious, he could try to execute random commands on a door in an attempt to disrupt its operation and put it out of sync with the key management system. His false card could supply random data posing as commands, in the hopes that it decrypts to something valid. However, each command must have the destination door number included in it, in addition to a valid opcode and proper format for the command data. The ratio of valid commands to possible plaintext strings is extremely low, and it would take Oscar years to successfully execute even one false command. Even if his false card was programmed to try millions of different ciphertext strings, he would probably be noticed if he stood suspiciously by a door for more than a few minutes.

Oscar creates a false card to supply false confirmations to the key management system

If Oscar could somehow gain access to the key management system, which is assumed to be physically secure to begin with, he might try to supply it with false confirmation message. However, confirmations are encrypted with the same key as used for commands, so the likelihood of him supplying a random

value that happens to decrypt to a real confirmation is as low as finding a valid command. In addition, he would certainly be noticed at the key management system after no more than a few minutes, as it is supposed to be physically secure.

Oscar creates a false key management system to supply false information to a card

Oscar's false key management system could try to give a legitimate card false commands, in an attempt to fill up the card's storage and provide false commands to a door. However, the system must supply the card with information such as a destination door, and the card's own identification number, all encrypted with K_{CS}. As with giving a false command to a door, an attempt to randomly guess a valid command ciphertext string will take years. The real card will certainly be noticed as missing by that time.

6.9.2 Attacks Not Prevented

Oscar attempts to extract information directly from the key management system

The key management system currently stores data, including encryption keys, in a file format that is easily read by an attacker. Thus, if Oscar has access to the central system, he will be able to access and modify its data. However, the central system is assumed to be physically secure; it does not need network access and can be placed in an isolated location, such as a safe, when not in use. A future version of the central system software may eliminate this security hole by encrypting the data it stores.

Oscar gets to a door before it's been initialized

Newly-manufactured doors are assigned the same door ID and K_{SD} (256 and the numerical value of a string of sixteen ASCII zeros, respectively). Oscar could potentially create a false card that initializes doors before the owner is able to, since he knows the door's ID and K_{SD}. In fact, this would be possible using any key management system software and a blank card. As a result, door should be initialized as soon as they are purchased. A future version of the system might allow each new door to have a random encryption key, which would have to be entered when the administrator's central system is instructed to set up a new door.

Chapter 7

Production Design

The Crypto-SmartLock was designed with a specific market in mind (described in Section 2.3 on page 18). Before it can be manufactured as a product, however, the design of the prototype must be altered slightly to be cost-effective and able to serve the needs of the customers. This chapter describes the design of the door lock, Smart Cards, and key management systems with respect to their mass-production.

7.1 Door

The prototype door we constructed contained components ideal for the development process, but that would not be suitable for a final mass-produced lock. Below is an analysis of the lock electronics and a design that utilizes components that would provide the necessary functionality at the lowest cost.

7.1.1 Components

The final door lock should be composed of only a MSP430, a smart card reader, a solenoid, and an LED, along with a handful of secondary components. As detailed in the MSP430 software design in Section 6.7.3 on page 68, the MSP430 must have 60K of on-chip flash memory; the MSP430F149 is the least-expensive chip that meets this requirement. It must interface with a Smart Card reader that operates on 3 to 3.6 volts, like the MSP430. The Phillips TDA8029 is essentially a smart card reader on a chip, and meets this requirement. It is designed for embedded applications, especially those that need to save on energy; its sleep mode needs only twenty μA. It requires only a physical contact module to interact with Smart Cards. Amphenol-Tuchel produces a physical contact module for exactly this purpose. The solenoid must be a bistable unit; BLP produces a latching solenoid suitable for this design. In order to interface with the digital outputs of the MSP430, the solenoid will need driver transistors to

allow the needed positive and negative currents to flow. Finally, Panasonic offers a red/green LED that our user interface calls for, at a reasonable price.

7.1.2 Power

Because the MSP430 usually operates at between 3 and 3.6 volts, and long-life lithium batteries supply 3.6 volts, we chose to standardize the components in the production system at 3.6 volts for power computations. With this in mind, the different components of the system draw the currents listed in Table 7.1.

Table 7.1: Current Draw of Components for Production System

Device	Current (mA)
MSP430F149 at 8MHz	2
TDA8029	50
LED	10
Solenoid	110
MSP430F149 (sleep mode)	0.0001
TDA8029 (sleep mode)	0.02

Based on this information, and the amount of time each component needs to be active, a timeline can be constructed of the current draw during a normal transaction. This is detailed in Table 7.2 and Figure 7.1. After inserting his card and being granted access, the user has a four-second window during which he can open a door. After those four seconds, the latching solenoid will return to the "locked" position, so that anyone following a legitimate user will not be able to enter the door.

Table 7.2: Current draw during unlock process

Time range (microseconds)	Current (mA)	Event
0-6	50	card reader starts up
6-225	52	MSP430 starts up
225-350	52	card reader ready to go
350-500	52	MSP430 executing code
500-600	52	code executed
600-150600	122	card reader off; solenoid and LED activated
150600-1000600	10	solenoid off; LED still on
1000600-4000600	2	LED off; MSP430 pauses
4000600-4150600	112	solenoid activated
4150600-4150606	2	solenoid off; MSP430 turns off

In addition to these, the system will draw 20.1 μA constantly, due to the draws of the MSP430 and card reader sleep modes. Note that command processing will also take some additional current, but this happens so rarely relative to normal door accesses that its effect is negligible. With this in mind, Table

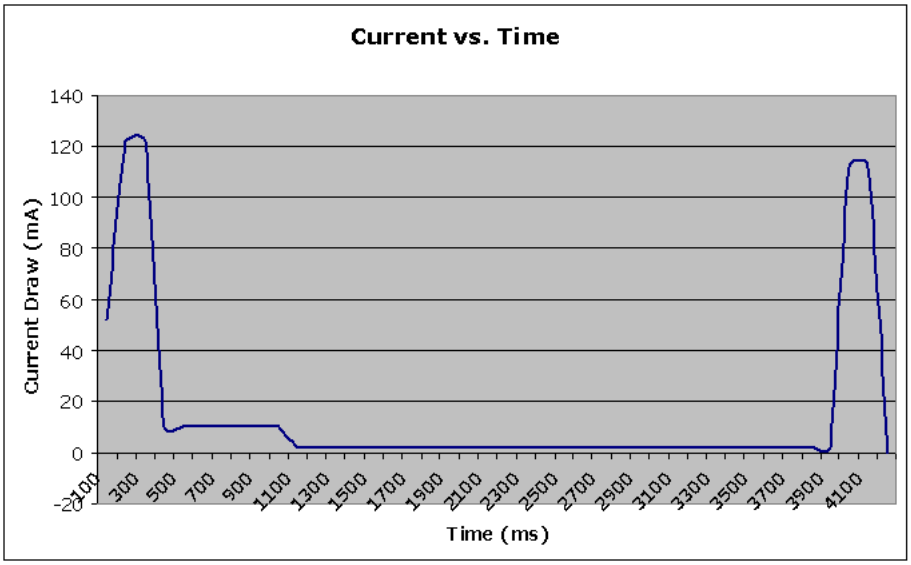

Figure 7.1: Current draw during unlock process

7.3 shows milliampere-hour values for the system, depending on how often a particular door is used.

Table 7.3: Power Usage

Access per day	mAh
10	0.043
20	0.067
30	0.091
40	0.115
50	0.139
60	0.163
70	0.186
80	0.210
90	0.234
100	0.258

We have decided that a one-year replacement cycle for the battery of the door is a reasonable minimum. With this in mind, 100 accesses a day will require a battery with 2.2 Ah of life. For comparison, Tadiran makes a 2.4 Ah AA cell, although mission-critical doors will need a wider margin of error, such as that supplied by Tadiran's 3.6 V 7.2 Ah C-sized battery. Other possible battery life values are shown in Figure 7.2.

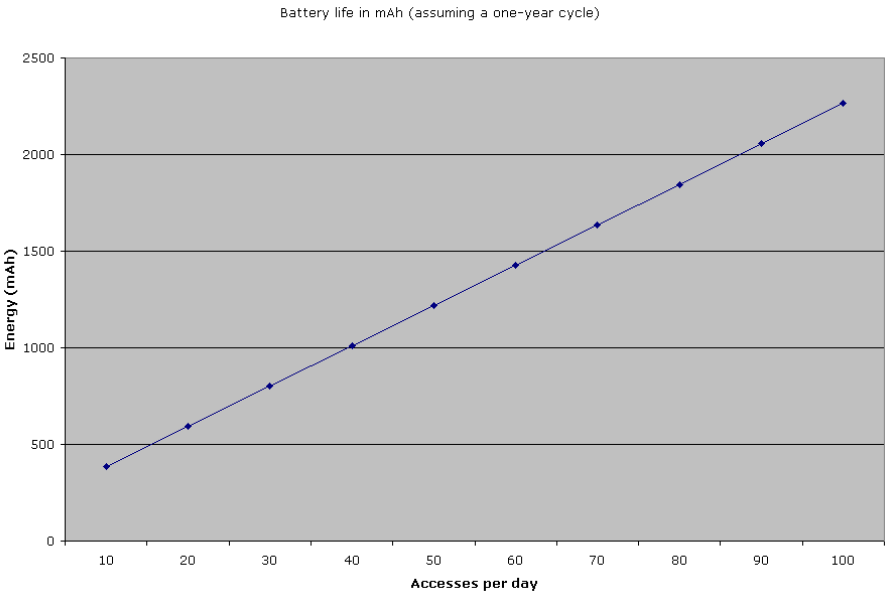

Figure 7.2: Power Requirements in mAh for a one year replacement cycle

7.1.3 Cost

The cost of a door lock to a consumer should be low enough that it is cost-effective in our target market, relative to the high security it provides. The production system uses the most inexpensive components possible that still allow for the full functionality of the system. An investigation of suppliers shows that purchasing 10,000 components at a time is the most cost-effective strategy, based on the bulk discounts advertised. Thus, the prices we collected are based on an initial production run of 10,000 door locks.

Table 7.4: Costs of Production Design

Device	Quantity	Cost per 10,000	Supplier
Philips TDA8029	1	$26,316	Avnet
TI MSP430F149	1	$67,200	Digikey
Panasonic P392 Red/Green LED	1	$20,377	Digikey
MMBF0202PLT1 P-channel Transistor	2	$13,100	Digikey
MMBF2201NT1 N-channel Transistor	2	$13,100	Digikey
BLP Solenoid	1	$86,620	RS Components
Amphenol-Tuchel C702 Card Contact	1	$79,625	Digikey
	Total	$195,376	

The cost of materials for each door lock is under $20. Note that this does not include the cost of a battery; a Tadiran 7.2 Ah C cell that supplies the proper voltage is $13.80, and is expected to last at least one year with a comfortable margin of error. These batteries can also be had for as low as $8.97 if purchased in bulk. Under a normal retail sales model, a door lock would cost approximately $80 as an "off the shelf" component. However, an organization would probably purchase many door locks at a time as part of a security upgrade initiative. In addition, the organization would also need to purchase the key management system software. Thus, a corporation marketing this system would probably sell the system as a package, which starts at a base price of perhaps hundreds of dollars and can be configured to include a set number of door locks, and a number of cards per door lock. Note that this analysis does not take into account non-electronic components which may already exist on the door, such as a handle or enclosure. Installing these components will incur additional costs, as will the adaptation of existing components from an earlier electronic system, primarily in the area of labor.

7.1.4 Block Diagram

The connections between the components of the door lock and the rest of the system are shown in Figure 7.3. This design represents a complete system ready for mass-production.

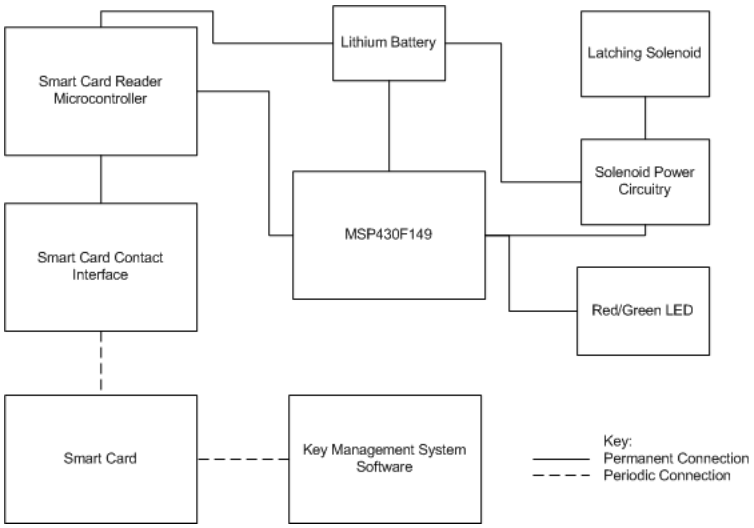

Figure 7.3: Production Design Block Diagram

7.2 BasicCard

A final production design may wish to revisit the memory requirements of the BasicCards. The ZC3.9 used in the prototype can support access to 133 doors if no commands need to be on the card. If no access keys are needed, the ZC3.9 can support 136 commands for a single door or up to 5 commands for each of 24 different doors.

While this is more than sufficient for our prototype, certain users of the final system may need more space on the card. Our proposed solution for the final system is to support two card types. The existing ZC3.9 card will be used for most users of the system. Specific users needing more storage capacity would be issued ZC5.5 Professional BasicCards. These users would include people who may need access to more than 133 doors and people such as janitorial staff who need to be able to carry many commands for many different doors.

7.3 Key Management System

For a final production design of the Key Management System, several details which were omitted in the prototype system would have to be taken under more careful consideration. For the prototype, some of the details of the system were intentionally watered down.

One of the items which was was left incomplete was revisioning during serialization. Currently if an administrator exports the database to some file, then modifies the system, and finally synchronizes the changes to a user's card, the exported database has become out-of-sync with the rest of the system and should be prohibited from import, but the system will allow the import to succeed. This will allow commands to be issued without the Key Management System being aware of the commands after the import. In the production design, a complete revisioning system would need to be added so that if any commands are sent to the card reader it would have to invalidate any outstanding backups of the system, or a command would need to be added to the doors for the Key Management System to request that the door send back a checksum of the current state of the lock. The first option would be the easiest to implement but would make it hard on administrators to keep backups of the system. The database would need to be exported after every change to the cards. Using this option would render export usable almost solely for moving the Key Management Software to a different computer. Using the second option, to have the door return it's state would only solve the problem whereby the commands sent to the card have already been executed on the door. If the command was still on the card, having the door return it's state would not prevent the commands from executing, but let an administrator know that the door was modified in a manner inconsistent with the Key Management Software.

Another important feature that would be in a production version of the Key Management Software would be further enhanced security. One way to accomplish this would be to encrypt the serialized data with a hashed passphrase.

This in addition to requiring a user to log in with a username and password would help to secure the Key Management System.

One more feature that was intentionally left out of the prototype design was the ability to grant users access by a selecting a door. Since the database does keep track of the required information to do this it would not be much to include in the production version this feature.

Chapter 8

Conclusion

The goals of our project were all accomplished successfully. Our final system as implemented provides excellent physical security using cryptography and Smart Cards. The key management system is capable of managing the many locks and cards that a medium-sized organization will require. Likewise, the door locks are able to process commands from the key management system to allow the dynamic access control expected from electronic locks, using only the Smart Cards as the communications medium. In addition, the innovative communications protocol prevents any attacks that may compromise security. The entire system is well on its way to being ready for the commercial marketplace.

8.1 Future Work

As with any academic project, time has prevented us from fully covering every aspect of the system, and thus the opportunity to expand this project should not be overlooked. Possibilities for future work include, but are not limited to, mathematically analyzing the protocol for cryptographic security, using ad-hoc wireless networking to connect the doors to the key management system, producing a prototype more in line with the production design, and changing parameters in the system such as the BasicCard or the MSP430 to incorporate a more easily expandable version of the project. Some of these possibilies are also detailed below. In addition, although the system we developed was specifically targeted at doors, there are many other uses for a cryptographic card authentication system. Some of the possibilities include automotive locks and ignitions, recording employee time-card information, and secure credit card transactions at point-of-sale locations.

8.1.1 Random Numbers

The random number generation in our key management software and door lock software is done by standard C libraries. Because the properties of these libraries

are well-known, they could represent a possible vulnerability. As research in random number generation continues, new methods might be discovered that can be applied to this project. For example, a non-deterministic devices, such as a variable resistor or diode, could be added to the 430 to provide a source of random numbers. Likewise, cryptographically-secure algorithms that generate pseudo-random values could be implemented for the key management system.

8.1.2 Smart Card Evolution

Advances in Smart Card technology could be applied to this project as they come about. For example, contactless Smart Cards that use short-range radio communication are becoming commonplace. As these cards become more cost-effective, and any security concerns surrounding them are alleviated, they could be integrated into the existing Crypto-SmartLock framework. In addition, Smart Cards do not have to take the form of an actual card, but could instead be put into a different package. One possibility could be a metal key-shaped "Smart Card," which would have the sturdiness of traditional mechanical keys, and would also seem more familiar to people unaccustomed to electronic locks.

Appendix A

User Manual

A.1 Crypto-SmartLock System Requirements

The Crypto-SmartLock system allows the addition and management of a multitude of users and door locks. To run the system, one door lock module is required for each door that needs to be secured, and one BasicCard formatted for use with Crypto-SmartLock is required to serve as a key card for each user that will be given access to doors in the system. In addition, a copy of the Crypto-SmartLock Key Management System software is required.

This system is based around a disconnected, centralized Key Management System. That means that every door lock module is a battery-powered standalone unit that is not connected back to the Key Management System with physical wires. Instead, the same key cards that give users access to doors are used to configure the door lock modules.

Before any door lock modules are installed, a copy of the Key Management System software must be installed and configured on an IBM-compatible PC running Microsoft Windows 2000 or later. The Key Management System software requires a PC/SC compatible Smart Card reader to be connected to the PC. Most personal computer Smart Card readers on the market today come with PC/SC compatible drivers, check their documentation to make sure.

To ensure the security of the Crypto-SmartLock system, the Key Management System should only be installed on a single computer, and this computer must be secured. It is recommended that the Key Management System computer be kept in a secure location, and that login access to the system be carefully managed.

A.2 Basic Key Management System Operation

All functionality in the Key Management System is on six tabs: *Manage Users*, *Manage Doors*, *Manage Access*, *Reissue Commands*, *Assign Commands*, and *Card Reader*, as seen in Figure A.1. The first task in setting up a new system is

adding users. Before any door lock modules can be setup and installed, at least one user must be added to the system.

A.2.1 Adding Records of Users

Users may be added to the system in the *Manage Users* tab. To add a new user, click on the *New User* button. A new entry will be shown in the list of users on the left side of the tab. Up to 1024 users may be added to the system in this manner. The user's name should then be entered in the *Name* textbox. Once the name is set, click the *Apply* button next to the textbox to save the new name. A user's name may be changed at any time by selecting their entry from the list at the left, then modifying the name in the *Name* textbox, and clicking the *Apply* button. Keep in mind that the system internally uses the user's ID number to manage access to doors, so the name value will not affect access settings per user ID. It is advisable to use unique user names whenever possible, to avoid confusing users. No differentiation is given to users that may be administrative staff in the system. Commands to setup or modify the state of a door lock module are not assigned to any users by default.

A.2.2 Adding Records of Doors

Records of doors are added to the Key Management System just like users are added. This is done in the *Manage Doors* tab. Each new door may be added by clicking the *New Door* button, which will add an entry to the list at the left.

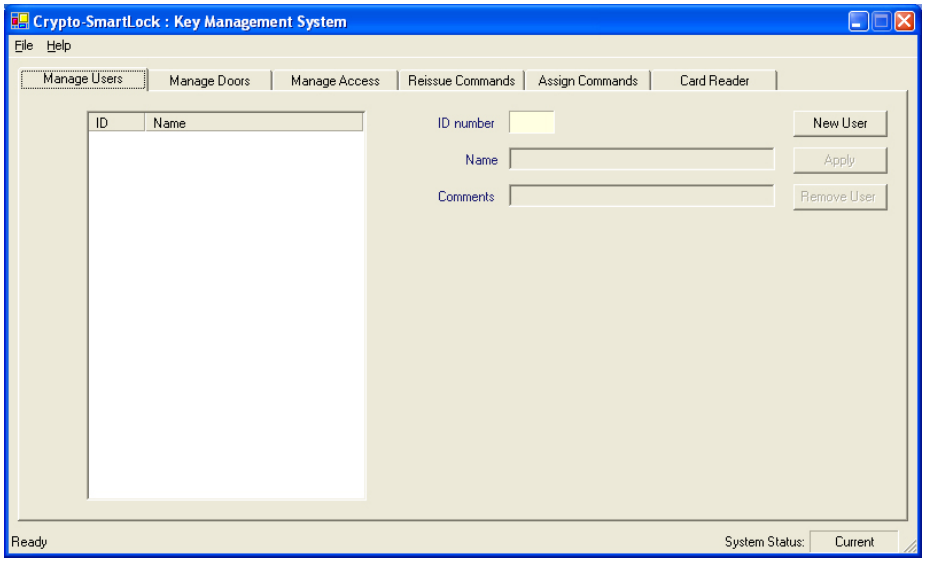

Figure A.1: The Crypto-SmartLock Key Management System

For each door added, its location should be denoted. Change the text in the *Location* field, and click the *Apply* button. Like a user's name, the door's location should be a unique identifier that is meaningful to the administrators of the Key Management System. The system internally uses the door's ID number to keep track of access permissions, so the location value may be modified without affecting the state of the door.

A.2.3 Granting and Denying Users Access to Doors

Each time a user is added to the system, they start without access to any doors. Access may be easily added and changed around within the *Manage Access* tab. This tab allows a system administrator to change access permissions on a per-user basis. A user is first selected from the *Users* list at the left. Whenever a user is selected, the *Doors* list at the right will automatically update, showing which doors the selected user currently has access to. Access is denoted with checkboxes: if a door shows a check-mark, the selected user has access to it, otherwise the user does not. These permissions may be changed by clicking each checkbox to set or clear a check-mark. Whenever the current access permissions are changed, the door affected will be highlighted in red. To save any changes made, click the *Apply* button. This version of the Key Management System software does not provide for changing access permissions on a per-door basis - e.g. selecting a door and changing which users have access to it.

A.2.4 Updating Door Lock Modules

Since this is a disconnected, centralized system, doors are not connected to the Key Management System via physical wires, so they do not receive updates about changes to the system in realtime. Instead, each door receives changes in the form of *Commands*, which are transferred on a user's key card. Appropriate commands are automatically generated by the Key Management System whenever users are added or removed, doors are added or removed, or access permissions are changed. To get these commands to a door, an administrator must assign them to one or more users. By default, the only type of command that is automatically assigned to a user is the command to add their access to a door.

Any command that has not yet been assigned to a user is stored internally in the *Unconfirmed Commands Queue*. On the *Assign Commands* tab, these commands may be seen in the list at the right labeled *Unconfirmed Commands*. To assign them to a user, first select a user from the *User List* at the left. Next, select a command from the *Unconfirmed Commands* list on the right. Click the *Assign Command to this User* button, and the command will be scheduled for addition to that user's card. The *Copy Command to this User* button may be used if an administrator wishes to assign a command to more than one user, but this is usually not necessary. To remove a command from a user, select the command from the *Commands assigned to User's Card* list, and click the

Remove Command from this User button. The command will be moved over to the *Unconfirmed Commands* list, where it may be reassigned to any user.

A special case of commands is the REMOVE USER command, which is issued when a user is denied access to a door, and the user already had an ADD USER command copied to a card to grant them access. Along with each REMOVE USER command, there is a PLACEHOLDER command that must go along with it. Future versions of this software would not show the PLACEHOLDER, it would be automatically associated with any REMOVE USER command. The current version, though, requires manual copying of a PLACEHOLDER along with any REMOVE USER command onto the same card. The PLACEHOLDER is required because it denies the user's access from being re-added to a door with an old command once it has been removed. Each REMOVE USER and PLACEHOLDER command pair should be assigned to one or more users, and these user's key cards synchronized with the new commands. Once all of the commands have been processed by the appropriate doors, the removed user will no longer have access to each door that they were removed from.

A.2.5 Putting Commands on Key Cards

Once some commands are assigned to users in the Key Management System, they may be put on key cards. The Crypto-SmartLock system operates on a principle of the Key Management System, all door lock modules, and all key cards being synchronized with each other. That is, all components of the system agree with each other about the system's state. This principle leads to the concept of *synchronizing* a key card with the Key Management System. It is not simply a matter of copying commands to the card - once information is transferred, the Key Management System internally marks any commands transferred to a key card as being *unconfirmed*. This means that these commands have been placed on a card, but that the system has not yet received confirmation that the commands were successfully processed by their destination door lock. Once a door lock module successfully processes the command, it will write a *Confirmation* to the card for the Key Management System to receive upon the next synchronization with the card.

All communication with key cards occurs on the *Card Reader* tab of the Key Management System. With a PC/SC compatible Smart Card reader installed and configured, insert a key card into the reader, and click on the *Initialize Card Reader* button. Make sure a card is never removed in the middle of an operation, e.g. when the status light on the card reader is lit. If something goes wrong, a status message will pop up explaining the problem. If all goes well, information on the tab will update, showing what type of card has been inserted. If the card inserted has not been assigned to a user yet, it will be shown as a *Blank Card*. In this case, a list of *Users not assigned a card* will be enabled in the bottom left of the tab. Select a user to assign to the card, and click the *Assign User to This Card* button below the list. The screen will update with the new user ID and name assigned to the card.

At this point, any commands that are assigned to this user will show up in a list at the bottom middle of the tab, entitled *Commands to be sent to User's Card*. These cannot be modified on this tab, they are only shown for reference. To send these commands to the card and check for any confirmations coming back, click the *Synchronize with Card* button. Do not remove the key card from the card reader until the tab indicates a *Synchronization Complete* message. This message is shown under the *Number of Confirmations* field when communication with a key card is complete.

If any confirmations were returned to the system from a key card during synchronization, they will now be shown in the *Command Confirmations Received* list. These indicate that a command was executed successfully on a door lock module.

A.3 Door Lock Modules

Figure A.2: The Crypto-SmartLock Prototype Door Module

Shown in Figure A.2 is a prototype for a Crypto-SmartLock door lock module. This picture shows a key card inserted into the integrated Smart Card reader. The door lock modules are very easy to operate, a user only needs to insert their card. The door module provides visual feedback via a single Light

Table A.1: Door Lock Human Interface Design

Event	**LED Indication**
Processing Commands	Light Orange until processing complete
Command Error	Flash Red and Orange alternatively for 1/4 second each, and two seconds total
Access Denied	Blink Red on and off for 1/2 second each, three times
Access Granted	Light Green for one second

Emitting Diode (LED). Possible LED patterns are shown in Table A.1. If there are any commands on a card for the door to process, the LED will indicate that the door is processing commands. Once that is complete, the LED will indicate whether access to open the door has been granted or denied.

A.3.1 Setting up a Door Lock Module for the First Time

New door lock modules, like new key cards, must be instantiated before they can be used in the system. Like everything else involving communication between the Key Management System and a door lock, this is done through the use of commands transferred on a key card. When a new door is added in the Key Management System, a command called SETUP DOOR is added to the list of *Unassigned Commands*. Each SETUP DOOR command applies to a single door. The Key Management System lists the door ID that each SETUP DOOR command would instantiate a door to. Future revisions of the software would also list the corresponding location for ease of use, but the current version does not.

To set up a door lock module, first decide which door ID to assign to the new door lock. Next, in the *Assign Commands* tab, select a user to assign a SETUP DOOR command to. Select the SETUP DOOR command in the *Unassigned Commands* list with the correct door id (denoted in the *DID* field). Click the *Assign Command to this User* button. Note that the system will not allow more than one SETUP DOOR command to be copied to a single card at one time. This is because it could cause confusion as to which door ID would be assigned to the first door that ran a command, and each other door after it.

Insert the user's key card that the SETUP DOOR command was assigned to, and synchronize with it in the *Card Reader* tab. Now, insert the card in a new door module. Once command processing is complete, that door is now initialized as the door ID indicated in the command. Note that after the command is processed, the door will try to grant or deny the user access to the door with its new door ID. The next time a key card with commands on it is inserted into the door lock, the door will only process commands that are meant for that door ID.

A.4 Removing Users and Doors from the System

The removal of a user may be a simple or fairly complicated procedure, depending on how much access a user already has to door lock modules in the system. If a user has not been granted access to any doors, then removing them from the system is as easy as selecting their name from the user list on the *Manage Users* tab, and clicking the *Remove User* button on the right. All record of the user vanishes from the Key Management System controls, but the system internally keeps a copy of all the user's information. Future versions of the software could allow for a user to be re-instantiated after a removal, but the current version does not.

If a user has been granted access permissions to one or more doors in the system, but none of the commands to grant access were copied onto key cards yet, then the removal of a user will result in the system automatically removing any commands pertaining to that user. Any commands that were assigned to the user being removed that have to do with other users in the system will automatically be moved into the *Unassigned Commands* list in the *Assign Commands* tab.

If any commands to grant access permissions have been added for a user to any key cards, whether they have been executed by a door lock module or not, REMOVE USER commands will be issued into the *Unassigned Commands* list for each one. Along with each REMOVE USER command, there is a PLACEHOLDER command that must go along with it. The reasoning for this is explained in the *Updating Door Lock Modules* section above. Each REMOVE USER and PLACEHOLDER command pair should be assigned to one or more users, and these user's key cards synchronized with the new commands. Once all of the commands have been processed by the appropriate doors, the removed user will no longer have access.

Removing doors from the system is always a simple procedure. Simply select the desired door from the *Manage Doors* tab, and click the *Remove Door* button. Do not remove the wrong door, however, because the current version of the software does NOT support re-instantiating doors that have been mistakenly removed.

A.5 Reissuing Lost Commands

If it happens that a command is issued to a user's key card, but something happens to that key card, an administrator may wish to reissue the same command to someone else. In this case, they may do so by using the *Reissue Commands* tab. Only commands that have been previously copied onto a user's key card, but not confirmed, are available to be reissued. Simply select a command from the *Unconfirmed Commands* list at the left, and click the *Reissue this command* button. The command will be moved back into the *Unassigned Commands*

list. From there, it may be assigned or copied to any user's card in the *Assign Commands* tab.

Appendix B

Table of Contents for CD

File	Description
README.txt	CD Index
Basic Card Software	Software for the BasicCard
Drivers	All drivers and utilities used
Documents	All reports and LaTeX source code
Door Software	Software for the MSP430
KMS Documentation	HTML Documentation for PC software
Web Site	The project website
MQP.pdf	The final report for the project

Appendix C

Bill of Materials

Item	Vendor	Quantity	Price	Total	Paid By
MSP-449STK	SparkFun	1	65.95	65.95	Frank
MSP-1121STK	SparkFun	1	19.95	19.95	Frank
MSP-JTAG	SparkFun	1	11.95	11.95	Frank
Shipping	SparkFun	1	5.38	5.38	Frank
ZC3.9	ZeitControl	10	4.60	46.00	Jeff
Shipping	ZeitControl	1	35.00	35.00	Jeff
PC Board	RadioShack	1	1.69	1.69	Frank
Power Jack	RadioShack	1	2.49	2.49	Frank
Smart Card Kit	ACS	1	99.00	99.00	Prof. Duckworth
LM317	ECE Shop	1	.75	.75	Shop Account
MC2805	ECE Shop	1	.50	.50	Shop Account
Battery Cases	ECE Shop	3	1.5	4.5	Shop Account
Plastic Case	ECE Shop	1	10.00	10.00	Shop Account
Green LED	ECE Shop	1	.50	.50	Shop Account
Red LED	ECE Shop	1	.50	.50	Shop Account
Green/Red LED	ECE Shop	1	1.00	1.00	Shop Account
Total				305.16	

Appendix D

Development Tools

D.1 MSP430 Compilers

We initially planed to use the Gnu GCC as our C compiler. GCC's big advantage is that it is free and unlimited. Also, we have experience using GCC to develop for UNIX on various platforms and for PalmOS on the Motorola 68k. According to online sources, the quality of GCC is comparable to the far more expensive options from the commercial compiler vendors. We had difficulties getting GCC to compile the code examples from our development board vendor, and were unsuccessful in our attempts to use the graphical debugger. We turned to the demo version of the Quadravox compiler.

Table D.1: MSP430 Compilers

Compiler	IAR limited	IAR full	Quadravox	CrossWorks	Gnu GCC-MSP430
Price	free	$2500	$150	£99	free
Limit	2k code	none	F4[34]x family only	none	none
Debugger	GUI	GUI	GUI	GUI	CLI + optional GUI frontends

D.2 BasicCard

The BasicCard chosen for our development system is the ZC3.9. This card provides 8 kilobytes of EEPROM for program and data storage and 256 bytes of RAM.

The BasicCard is programmed using the specialized CardBasic language. All of the necessary tools and documentation to program the BasicCards is provided for free by ZeitControl. This software is capable of loading program code onto the BasicCard via a standard PC/SC compliant Smart Card reader, such as those in our Smart Card development kit.

D.3 Visual Studio®.NET 2003

The development environment chosen for the Key Management System was Microsoft Visual Studio®.NET 2003. Our initial options for creation of the Graphical User Interface (GUI) were to program in the C language using the WIN32-API, to use C++ with the Microsoft Foundation Classes (MFC), or to use Microsoft's Managed C++ extensions After careful consideration of all options, we decided on the Managed C++ extensions Using this option required porting the current back-end codebase created with Microsoft Visual Studio®v.6.0 to the Managed environment used in Microsoft Visual Studio®.NET 2003.

Microsoft's Managed C++ Extensions build off Microsoft Visual C++®and add such features as garbage collection, and byte code compatibility with other Managed Languages.

Visual Studio®.NET 2003 is Microsoft's primer integrated development environment. It has support for producing professional looking Windows Graphical User Interface applications using drag-and-drop forms wizards. Using Microsoft Visual Studio®.NET 2003 allows rapid prototyping of the Graphical elements in the Key Management System while enabling us to focus our attention on more critical development tasks.

Appendix E

Internal Administrative Tasks

E.1 Web Site

A web site was set up as a mode of communication between the advisors and the project group. It is home to weekly progress reports and meeting notes as well as all of our documentation and research links. It is located at http://gw.jartech.net/mqp/, and will remain online as a repository of project materials.

E.2 CVS

We used the Concurrent Version System (CVS) to manage all files generated by the project. This includes the web site and project documents. CVS was configured so that all changes to the web site are immediately reflected on the web.

Bibliography

[1] Specification for the advanced encryption standard (aes). Federal Information Processing Standards Publication 197, 2001.

[2] Messerschmitt Systems AG. Corporate Web Site `http://www.messerschmitt.com/`, 2003.

[3] Atmel. Corporate Web Site `http://www.atmel.com/products/`, 2003.

[4] Mike Speciner Charlie Kaufman, Radia Perman. *Network Security: Private Communication in Public World*. Prentice Hall, second edition, 2002.

[5] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002.

[6] National Institute for Standards and Technology. Web Site `http://csrc.nist.gov/CryptoToolkit/aes/aesfact.html`.

[7] ID Enhancements Inc. Corporate Web Site `http://www.id-enhancements.com/iclass.html`, 2003.

[8] Texas Instruments. Corporate Web Site `http://focus.ti.com/mcu/docs/overview.tsp?familyId=342\&templateId=5246\&navigationId=11466\&path=templatedata/cm/mcuovw/data/msp430_ovw\&DCMP=TIHomeTracking\&HQS=Other+OT+home_p_micro430`, 2003.

[9] ADEL Lock. Corporate Web Site `http://www.adellock.com/english/`, 2003.

[10] Maxking. Corporate Web Site `http://www.maxking.co.uk/doorlockinginfo.htm`, 2003.

[11] Arizona Microchip. Corporate Web Site `http://asp.microchip.com/wwwParamChart/chart.aspx?branchID=1002\&mid=\&gdir=1010`, 2003.

[12] Motorola. Corporate Web Site `http://e-www.motorola.com/webapp/sps/site/taxonomy.jsp?nodeId=03t3ZGpnLn8636K100`, 2003.

[13] Sean O'Connor. Personal Interview, 2003.

[14] Vincent Rijmen. Web Site `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/`, 2003.

[15] Rabbit Semiconductor. Corporate Web Site `http://www.rabbitsemiconductor.com/products/rab2000/`, 2003.

[16] William Stallings. *Network Security Essentials: Applications and Standards: Second Ed.* Prentice Hall, 2003.

[17] Best Access Systems. Corporate Web Site `http://www.bestaccess.com/basisv.htm`, 2003.

[18] Bosch Security Systems. Corporate Web Site `http://www.boschsecurity.us/`, 2003.

[19] TimeLox. Corporate Web Site `http://www.timelox.com/`, 2003.

[20] The Silicon Trust. Web Site `http://www.silicontrust.com/trends/tr_healthcare.asp`, 2003.

[21] VingCard. Corporate Web Site `http://www.vingcard.com/page?id=480`, 2003.

Index

Access Permissions, 49
ACOS1, 32
ACR-30S, 65
Ad-hoc Wireless Networking, 87
Add Key to Card Command, 56
Add User Command, 56
AdelLock, 13
AES, 8, 19, 34, 44, 49, 58, 67
    ECB, 67
Apply Button, 49
Assign Command to User, 51
Assign Commands, 47
Atmel, 35
Attacks, 75
Authentication, 19

Backup, 22
Basic, 33
BasicCard, 33, 46, 54, 57, 99
Best Basis V, 14
Binary Exponential Back-Off, 24
Blank Card, 51
Blank Smart Card, 32
Bosch, 12
Brute-Force Attack, 44
Brute-Force Key Search, 20
Bus, 29

Card ID, 37
Card Reader, 47, 56
CardBasic, 33, 99
Centralized System, 10, 27
Challenge-Response, 20, 23, 30, 32, 44
Ciphertext, 22
ciphertext, 19
CLA, 57, 58
Classes
    Door, 53, 56
    Interface, 53
    Queue, 53, 55
    Reader, 53, 57
    User, 53, 56
Command, 38, 45
Command Generation, 55
Command Header, 39, 43
Command Transfer, 51
Communication, 40
Communication Door-Smart Card, 44
Communication Toolkit, 68
Communications Protocol, 37
Concurrent Version System, 101
Confidentiality, 19
Confirmation, 39, 45, 50, 51, 60, 67
Confirmation Header, 39, 43
Connected System, 26
Cryptographic Key, 19
Cryptography, 8, 19, 22
    Private Key, 19
    Symmetric-Key, 19
Cryptosystem, 19
Current Draw, 81

Data Integrity, 31
Database, 54, 56
Decentralized System, 27
DES, 19, 34
Disconnected System, 26
Door Alteration, 22
Door ID, 38
Door Lock Assembly, 9

encBlock1, 60

encBlock2, 60
Encryption, 40
Event Processing Loop, 67
Expiration, 26
Export, 55

Failure Points, 22
Flash, 36
Flash Memory, 67, 68

GCC, 99
Gilbert Vernam, 19
Graphical User Interface, 47, 54, 56, 100
    Tabs, 47

Human Interface Design, 65

ID iClass, 12
Import, 55
Infrared (IR), 30
Initialization, 26
Initialize Card Reader, 51
INS, 57, 58
Integrity, 19
ISO-7816, 31

JTAG, 63

Key, 9, 19
    Authorization, 38, 49, 56
    Card-System, 38, 56
    System-Door, 38, 56
Key Management, 10, 22
Key Management System, 23, 46, 57, 100
    Hardware Requirements, 46
    Class Structure, 53
Key Removal, 26

LaTeX, 1
LED, 67
Line-of-Sight (LOS), 30
Logging, 18

Manage Access, 47
Manage Doors, 47
Manage Users, 47
Managed C++, 100
Master Key, 25
MaxKing MaxLock, 13
Mechanical Lock, 8
Messerschmitt, 11
Microcontroller, 34
Microprocessor, 34
Microsoft Foundation Classes, 100
Microsoft Visual Studio .NET 2003, 47, 100
Monitoring, 18, 29
Motion Detection, 18
Motorola, 35
MSP430, 35, 36, 65

No Operation, 56
Nonce, 39, 45, 57, 60
    Command, 39
    Confirmation, 39
    Header, 39

Opcode, 38
Oscar, 75

Passphrase, 31, 85
PC/SC, 99
PDA, 14
Per User Cost, 22
Persistant Storage, 55
Persistant Storage Techniques, 85
Personal Identification Number (PIN), 24, 31
Physical Contacts, 29
Physical Removal of Door Lock, 48
Physical Security, 44
PIC, 36
Placeholder, 56
Plaintext, 19
Pre-Shared Keys, 25
Private Key, 22
Protocol, 19
PS/2, 65
Pseudo-random, 88

Quadravox, 99

Rabbit, 35
Radio Frequency Communication, 30
Random Numbers, 87
Reissue Commands, 47, 50
Remove User Command, 56
Remove Users, 48
Replacement Cycle, 82
Replay Attack, 20, 45
Restore, 22
Revert Changes, 47

Save Changes, 47
Sentinel Key, 38
Setup Door Command, 56
Shift Cipher, 8
Smart Card, 8, 23, 31
SparkFun, 65
Synchronization, 47, 50, 51

Tadiran, 82
TimeLox, 14
Training, 22
Transaction ID, 38, 45

Usability, 21
User Names, 48

Vernam Cipher, 19
Vincent Rijmen, 67
VingCard, 14

WIN32-API, 100
Worcester Polytechnic Institute, 36, 48

XOR, 19

ZeitControl, 33, 99
    Enhanced BasicCard, 34, 58, 85, 99
    Professional BasicCard, 33, 58, 85